Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

Jahan Rahmanova

MASTER THESIS

POSITIVE FEEDBACK IN SOFTWARE ENGINEERING

Submitted on 30.11.2022

Supervisor: Julia Krause, M. Sc; Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

_____

ERLANGEN, [DATE]

License

_____

ERLANGEN, [DATE]

Abstract

Continuous improvement of products and processes is one of the key goals of any software development team striving for success in its company. As we know from the practice, necessary improvements are not only hard to identify but sometimes it is even harder to integrate them into a working routine, they are met with hidden, unconscious, and sometimes even open resistance. The reasons for the resistance mainly originated from negative behavioral patterns, such as bad habits, past mistakes, negative work environment, unresolved conflicts, etc. To promote continuity of the improvements and keeping obtained progress, control mechanisms should be performed. Feedback serves as a communication channel to convey the output from the performed control mechanisms and the main goal of feedback is to improve performance.

In industrial practice, it is more common to provide negative feedback rather than positive. Good stable performance and a positive atmosphere in a team are more commonly taken for granted. As the German proverb says "Nicht kritisiert ist genug gelobt", which has a meaning „if you haven't been criticized then it's already a praise", which demonstrates a habitual perception of lack of feedback as positive feedback.

Positive feedback can help to avoid resistance against continuous improvement, as well as it can develop a good habit for repetitive good performance. Literature review and interviews have demonstrated that positive feedback has a positive impact on individuals and on the team itself, by motivating team members, making them confident in self-efficacy on their behold position, more satisfied with their job, and in general, happier with their life. Steve Jobs, who is one of the most influential people in the history of IT, said "It's not a faith in technology. It's faith in people.", he truly believed that great success and revolutionary changes in the world can be achieved by prioritizing people rather than technologies.

In the research, we have explored positive feedback in software engineering, factors that impact on the individuals' perception of the received feedback, as well as the human aspects that impact on the feedback evaluation. We have also found best practices, metrics that help to achieve high-quality product, and therefore, they promote to receive positive feedback by well-written requirements, efficient design, source code, testing. We also suggest to adopt positive feedback mechanisms in the software development process to boost productivity and performance of the team.

Contents

List of Figures

List of Tables

1 Introduction

This section describes the thesis motivation and goals that were determined prior to the research.

1.1 Thesis motivation

Thesis motivation and its goals were prompted by the data analysis from the interviews which were performed for a past research study.

In software engineering, continuous improvement of products and processes is maybe the most cited goal that professional teams are aiming for. In current industrial practice, improvement is a very difficult procedure, often restrained by open or hidden resistance. We assume, this resistance results from the fact that change is driven by mistakes and bad habits and therefore is always motivated by negative facts. Even the normal workflow is usually supported by various control mechanisms, like spell checker, syntax checker, review checklists, etc. that provide feedback directly when a mistake is made.

As the German proverb says “Nicht kritisiert ist genug gelobt”, which has a meaning „if you haven’t been criticized then it’s already a praise”, which should no longer be taken as habitual perception. A positive driver for change could transform the pattern of continuous improvement. Positive feedback and the fundamental change driver of “repeating good things” is used only scarcely and in an unsystematic fashion. Positive feedback should be exchanged during retrospectives and collected from users or customers when a product has satisfied their expectations.

To provide an overview of the studies, systematic literature review (SLR) about the positive feedback in software engineering will be conducted by following Kitchenham’s guideline. Also, SLR should cover the following research questions:

RQ1: What are metrics to provide positive feedback in software engineering?
RQ2: How to integrate positive feedback into the whole process of software development?

Moreover, interviews from the previous research study will be used as the practical evidence to confirm findings or discover contradictions with analysis from literature review.

The expected results will be an overview about metrics to provide positive feedback in software engineering, and an overview about how to integrate positive feedback into the software development process.

1.2 Update in the thesis goals

During the systematic literature review process, for the RQ1 we have explored not only best practices, metrics for the evaluation of performance to provide positive feedback, but we also suggest considering human factors that impact the evaluation of the performance to provide feedback (Ch. 3.2), as well as the feedback characteristics, and other human aspects that impact

on an individual's perception of the positive feedback. The rest of the thesis goals hasn't been changed.

1.3 Structure of the thesis

Chapter 1 of the thesis presents an introduction to the topic, update to the thesis and its structure.

Chapter 2 of the thesis presents the research methods that were used for conducting systematic literature review and interviews that were performed with practitioners from the industry.

Chapter 3 presents the research results, and it consists of the following subchapters:

3.1 impact of positive feedback on human aspects – observes factors and feedback characteristics that can impact on individuals' perception of the feedback.

3.2 code review as a feedback mechanism – overviews code review as feedback mechanism that get affected by human aspects, thus the feedback can be distorted.

3.3 interview analysis – results of the interview analysis inspired the directions of the research, and they were also compared with the results from the SLR.

3.4 the metrics, best practices to provide positive feedback in SDLC – responds on the first research question.

3.5 positive feedback from users – observes an impact of user feedback on the team, product, and company; presents an IS model where user satisfaction is one of the main metrics of the project success.

3.6 positive feedback mechanisms to integrate into the software development process – responds on the second research question.

2 Research

This section provides an overview of the research method that was used for conducting and analyzing interviews that were performed with practitioners from the industry prior to the research and for conducting the systematic literature review.

This research was motivated by the data analysis from the interviews which were conducted for a previous research study that was aiming to get practical insights about the creation and evolution of complete, consistent, and traceable requirements specifications.

2.1 Research Method

The method of this study is qualitative exploratory research that considers positive feedback in software engineering from different perspectives.

In our research, a systematic literature review is applied to evaluate and interpret research for a field of interest. Kitchenham's guidelines for performing Systematic Literature Reviews in Software Engineering[Kitchenham, 2004] are designed to provide a fair assessment of research topics using a robust, rigorous, and verifiable methodology. To prevent selection bias, a research protocol was created, where the following stages were described: background information of the study, research questions, defining search strategy (selecting data sources and search strings), defining research selection and quality criteria, defining data extraction process.

Feedback, a self-regulating mechanism, has existed from ancient times, and the first ideas about feedback occurred in the 18th century. The impact of positive feedback in software engineering has been already recognized in past research studies. Mainly, the impact of positive feedback was recognized on human aspects, as peer-to-peer feedback(horizontal feedback), and feedback from the manager(vertical feedback). However, the research about positive feedback in software engineering is very limited, especially in terms of positive feedback mechanisms and practices to integrate those mechanisms in a software development cycle.

Therefore, feedback was considered in terms of psychology and software engineering. By using keywords that were gathered from the interviews, literature was collected from reputable scientific resources, such as Google Scholar, IEEE Xplore, ACM Digital Library, Web of Science, and Springer Link. As there are not many research works regarding positive feedback mechanisms and practices in SWE, thus papers from well known journals and conferences were also used.

Qualitative and topic-specific criteria were formed to collect literature. To meet the qualitative criteria, literature must be written in English and analyzed through peer debriefing. Inclusion criteria for a specific topic were:

- Literature on positive feedback in software engineering;
- Literature on positive feedback mechanisms;
- Literature on the impact of the positive feedback during software development;
- Literature on the metrics to objectively evaluate good work results for positive feedback.

Literature that was focused on negative feedback in software engineering was excluded from the future study.

During the piloting of the research, valuable modifications were made to the inclusion criteria. Following criteria were added:

- Literature on general feedback mechanisms that motivate or initiate positive feedback;
- Literature on best practices to generate positive feedback;
- Literature on the impact of the positive feedback on the atmosphere in a team or organization as a whole;
- Literature on the benefits from the positive feedback to an individual and organization;
- Literature on software quality characteristics as metrics to provide positive feedback;
- Literature on user satisfaction as a metric of user positive feedback;
- Literature on factors that impact on individuals' perception of the feedback;
- Literature on sentiments in feedback that affect on feedback recipients' productivity;
- Literature on how to integrate positive feedback in the development lifecycle.

To validate inclusion and exclusion criteria, a pilot study with the sample size of 15 papers has been conducted, which helped to update inclusion criteria.

Research questions:

Originally, two research questions were set to be covered by SLR:

RQ1: What are metrics to provide positive feedback in software engineering?

RQ2: How to integrate positive feedback into the whole process of software development?

The goal was to explore positive feedback in software engineering from different perspectives. Therefore, we identified concepts related to the metrics, best practices to achieve positive feedback and positive feedback mechanisms that would be used during software development.

1) Since there are no specific metrics that indicate when positive feedback should be provided in software engineering, there were explored software quality characteristics, metrics and software artifacts that aim to achieve a high-quality software.

2) Positive feedback mechanisms were used to represent techniques that can be used during the software development process.

During the piloting of the research, we have found factors that impact individual's perception of the received feedback, so that positive feedback can have not only positive impact on an individual, but also it may have no effect or even a negative effect on the external and internal attributions of an individual's, such as performance, motivation, job satisfaction, behavior, and mood. In addition, the assessment of the source (the feedback sender) may potentially be impacted by factors that were discovered, making it possible that the positive feedback given by the impacted source is not unbiased.

Literature that was fitting to the inclusion criteria was analyzed with the MaxQDA data analysis tool. Peer debriefings were also conducted during our QDA in order to continuously assess and enhance the quality of our study. For data triangulation, semi-structured interview data from

the previous study was used to expand and supplement the results from the systematic literature review with the industry insights.

The semi-structured interviews were conducted with 13 practitioners from the industry. The participants of the interviews can be grouped into two categories: project-internals and consultants. “Project-internals” as requirements engineers and product owners are working with requirements within a specific project context. “Consultants” are often involved in different projects, so they are well-versed in different types of projects.

For collecting interviews, a qualitative survey method was selected. This method is based on the guidelines described by [Jansen, 2010], with the aim to get practical perception about the creation of complete, consistent, and traceable requirements. Jansen describes qualitative survey as a process that allows practitioners to provide extensive answers, which guarantee full and detailed coverage of the question, as in case for a past research: “What are the challenges and practices to create complete, consistent and traceable requirements specifications?”

Moreover, the transcripts of the semi-structured interviews were analyzed using qualitative data analysis (QDA). This analysis has motivated us to make research regarding the positive feedback in SWE. Out of 13 interviews, only 8 interviews indicated a positive feedback in software engineering.

To capture important information that were directly or at least slightly related to research, main codes were created. They were used as keywords or as ideas that were interpreted as keywords to gather information for systematic literature review.

The Results section presents analysis for each matching interview.

3 Results

Feedback is a widely used concept in software engineering that has multiple definitions depending on the context of its usage. The general definitions of the feedback [dictionary.com, 2022] are:

- a reaction to a particular process or activity.
- evaluative information derived from such a reaction or response.

In psychology, feedback is defined as knowledge of the results of any behavior, considered as influencing or modifying further performance [dictionary.com, 2022]. Feedback provides the individuals with the knowledge that they can use to assess how accurate their prior behavior corresponds to an "internal goal standard"[Bandura, 1986, 1991]. Feedback can be considered as a tool that can be used to navigate the work processes to effectively achieve desired goals.

In education, feedback is also used to guide learning efforts of individuals. However, feedback has also been reported to have detrimental effects on learners' motivation and emotions.

In business, feedback from the customers is known as a predictor of user's churn and retention.

Numerous studies about the feedback in psychology, in education and in business were devoted, however positive feedback especially in software engineering hasn't received due attention and recognition from the researchers. However, positive feedback has a strong positive impact on performance, motivation, self-efficacy, job satisfaction, mood of the individuals, which in turn affects the success of the project, atmosphere in the team, and the organization's productivity and reputation.

Positive feedback from the users has also a strong positive impact on the organization, product viability, and a software team. User feedback affects various dimensions of organization, from one side it promotes the popularity of the software product, at the same time it boosts self-efficacy of the team members, and enhances reputation and brand name of the company, and as the result, the revenue of the company also increases.

However, during our research study, we have found feedback characteristics and factors that affect an individual's perception of the received feedback, so that positive feedback can have not only positive impact on an individual, but also it may have no effect or even a negative effect on the external and internal attributions of an individual's, such as performance, self-efficacy, motivation, behavior, job satisfaction, and mood. In addition, there were found factors that may also affect the evaluation of the source (feedback sender), so that the positive feedback provided by the affected source may not be objective.

3.1 Impact of the positive feedback on human aspects

One of the earliest and most influential theories about feedback is Thorndike's Law of Effect. According to his theory, "satisfying state of affairs" is likely to be repeated, and behaviors with unpleasant outcomes may be stopped. Feedback serves here as a "connector" between response and prior stimuli [Kulhavy & Wager, 1993]. Later, based on Thorndike's theory Skinner has proposed a method of learning - Operant Conditioning, by which the effect or consequence of a response determines the likelihood of this response being repeated. Skinner's operant conditioning proposes 2 principles: "positive reinforcement" when behavior is encouraged with reward - "reinforcer", that reinforces the repetition of the desired behavior, and "negative reinforcement" works respectively the other way around [Buffardi & Edwards, 2014]. According to Skinner, "the external environment of the organization must be designed effectively and positively so as to motivate the employee" [Amutan, 2014].
Following from the Operant Conditioning principles, Kluger and DeNisi have concluded that positive and negative feedback interventions should improve performance because one reinforces right behavior and the other punishes wrong behavior [Kluger & DeNisi, 1996].

The software engineering community is mainly focused on the technological aspects of software development, but it overlooks the human factor, as organizational and social aspects, which play a significant role in the development process [Perry, 1994]. Lenberg [Lenberg, 2015] also noted the gaps in the field of research related to the human aspects of Software Engineering, therefore, inspired by Behavioral Economics, P. Lenberg proposed a definition of the research area Behavioral Software Engineering (BSE), which was defined as "the study of cognitive, behavioral and social aspects at different levels relating to the work of software engineers" [Lenberg et al, 2015]. Lenberg et al. suggested SE researchers explore behavior, thinking, and mood of the individuals that are "involved in software development and engineering" as they believe that this knowledge can be beneficial for the software engineers and for the end users of the software product[Lenberg et al, 2015].

One of the main research works for positive feedback in software engineering was made by Rien Sach. He reported that software engineers experienced feedback by evaluating the characteristics of the received feedback, which are influenced by software engineers' individual value set, as well as their mood at the time of receiving feedback, which affect their perceptions, assessments and individual value set [Sach, 2013]. Sach adds that feedback assessments may have an impact on motivation, performance, behavior, job satisfaction, and mood of software engineers [ibid: Sach, 2013].

Feedback is identified as a contributing factor in a variety of theories of motivation: Job Characteristics Theory (Hackman and Oldham, 1976), Goal Setting Theory (Locke, 1968) and Achievement Theory (McClelland, 1961) [Sach, 2013].
The following sections present the theories of motivation that include feedback as a factor that affect motivation, and the role of the feedback in the theories.

3.1.1 Goal setting theory

According to Locke's Goal-setting theory (1968), feedback provides individuals an understanding of how productive their performance was (knowledge of score), and some specific forms of feedback, such as "praise and reproof", have been recognized as performance-influencing factors [Locke, 1968].
Locke observed that knowledge of score received through feedback, can impact an individual's motivation, when an individual considers the feedback received to be valid and only when an individual can use information from the feedback received to set a new goal [Locke, 1968]. As explained when an individual receives feedback from someone who, according to his/her personal value set, is not considered to be in a position (e.g., hierarchy) to provide feedback, an individual doesn't perceive received feedback as valid feedback and it doesn't impact on an individual's motivation. Also, if the knowledge of score(feedback) was considered as valid, but it didn't consist of specific information for setting a new goal, then in this case, according to the Goal-setting theory, the feedback will not affect motivation. It has been proposed to use knowledge of score (feedback) to influence future goals and set desired performance standards. Locke also observed the correlation between "the level of the standard" in feedback and the "level of goals", e.g., when positive feedback was provided on a low-level standard performance, an individual's motivation was at a lower level compared to the situation when positive feedback was provided on a high-level standard performance [ibid: Locke, 1968].
Praise or reproof (feedback forms) has a changeable impact on motivation. Locke observed that praise (positive feedback) more efficiently improved overall performance, while reproof (negative feedback) improved performance when feedback was given in such a way that it can be compared to "a standard"[ibid: Locke, 1968].
Additionally, according to Locke's theory, competition is the main factor that motivates individuals to achieve higher levels of performance, since the performance of another individual serves as a model for them to compare their performance and to evaluate the success of their goals. Locke also believes that competition motivates individuals to innovate, as individuals mainly create innovation with the goal to improve performance through better practices, procedures, products, etc. [ibid: Locke, 1968].

3.1.2 Job characteristics theory

Hackman and Oldham in Job Characteristics Theory (1976), found that "core job characteristics" that are shown in Figure 1, have an impact on the following "psychological states", which are moderated by employee growth need strength:"

- Meaningfulness of work: The degree to which the individual experiences the job as one which is generally meaningful, valuable and worthwhile.
- Responsibility of outcomes: The degree to which an individual feels personally accountable and responsible for the results of the work he or she does.
- Knowledge of actual results: The degree to which the individual knows and understands on a continuous basis, how effectively he or she is performing the job."

[Hackman & Oldham, 1976]

These three psychological states constitute a "self-perpetuating cycle of positive work

motivation", which are powered by "self-generated rewards" (e.g., positive feedback). In its turn "self-generated rewards" enhance personal and work outcome, which are likely to continue as long as the individual values internal rewards that have been achieved through high performance, and also as long as all above-mentioned psychological states are present [Hackman & Oldham, 1976]. Individuals need to experience internal rewards, to regain internal rewards individuals will work harder and their productivity will be increased. However, this self-perpetuating cycle might be stopped, if an individual won't appreciate any longer the internal reward that was gained through high performance.
It is very crucial to let individuals be responsible for their work outcomes on a "meaningful task", however it is equally important for individuals to have "knowledge of the results", as without feedback individuals won't obtain internal rewards from their performance, which boost their motivation. Likewise, individuals who receive positive feedback but assess the task as trivial or who don't perceive themselves responsible for their outcomes of the work, their internal motivation won't increase [Hackman & Oldham, 1976].

Hackman and Oldham reported that jobs with high motivation potential will elicit favorable responses from individuals who have a strong need for personal growth and development ("growth need strength"), whereas the opposite is true for those who have a low need for growth. According to the Job characteristics theory an employee's growth need strength can have an effect at one of the following areas and sometimes at both:

1) between the job characteristics and the psychological states.
2) between the psychological states and the personal and work outcome.

As shown in Figure 1, Hackman and Oldham found that the psychological states lead to the following personal and work outcomes:
High internal work motivation.
High quality work.
High satisfaction with work.
Low absenteeism and turnover.
Multiple personal and work outcomes can get affected by psychological states.

Hackman and Oldham defined *feedback* as:
"The degree to which carrying out the work activities required by the job results in the individual obtaining direct and clear information about the effectiveness of his or her performance".

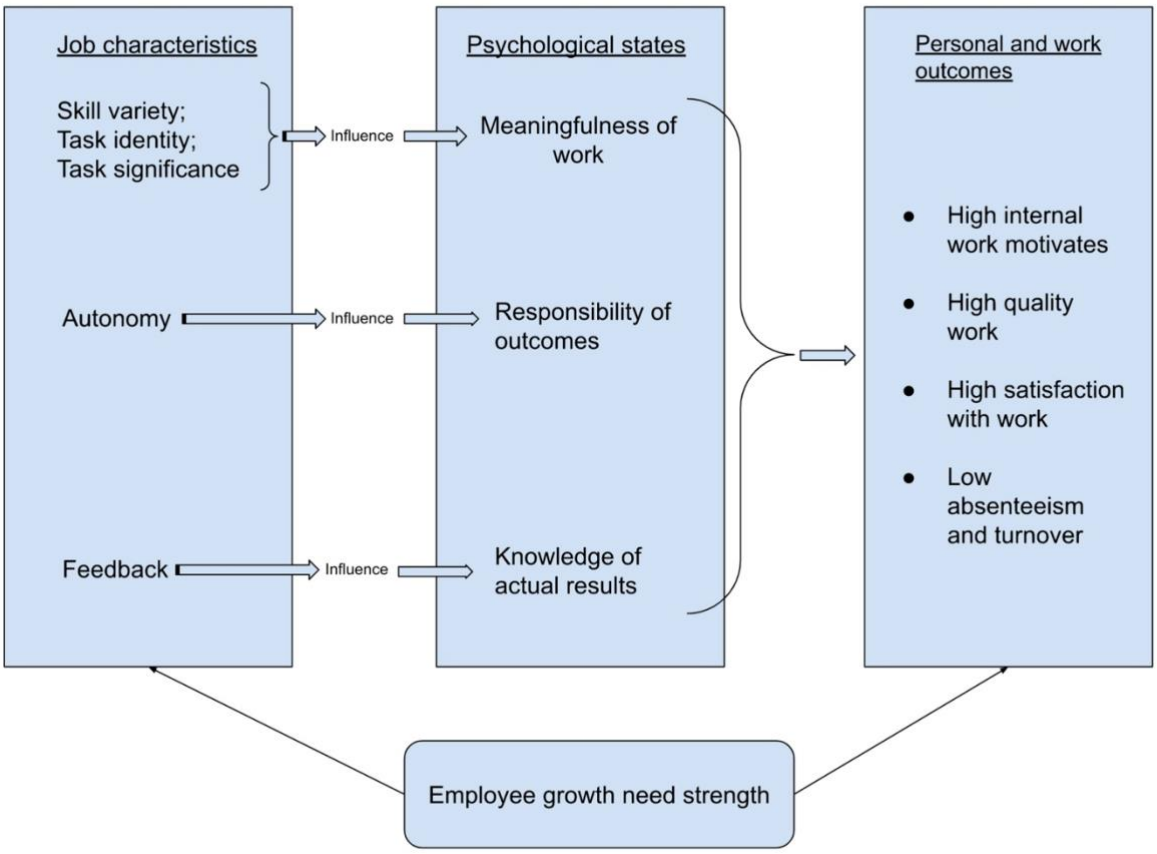

Figure 1: JOB CHARACTERISTICS THEORY MODEL OF MOTIVATION (Hackman and Oldham's Model)

Hackman and Oldham concluded that it is feasible to "measure the potential for a job to be motivating", as "the overall potential of a job" increases "internal work motivation" for employees to its highest, when all following conditions are true:

4 the job is high on at least one of the three job dimensions (skill variety, task identity, task significance) that contribute to experienced meaningfulness of work,
5 the job is high on autonomy that contributes to experienced responsibility for outcomes,
6 the job is high on feedback that contributes to experienced knowledge of results.

Hackman and Oldham presented Motivating Potential Score (MPS) measure, that is based on the above-mentioned conditions:

$$MPS = \left|\frac{Skill\ variety\ +\ Task\ identity\ +\ Task\ significance}{3}\right| \times Autonomy\ \times Feedback$$

Feedback in Job Characteristics theory is presented as one of the core job characteristics, that provides feedback recipients with the 'knowledge of the result' of their performance and also influences their psychological state, which is called ‘knowledge of actual results' of their work, and this psychological state is required for any job to be motivating. Thus, feedback, influencing psychological state, also impacts on personal and work outcomes.

3.1.3 Achievement theory

McClelland stated that human behavior is affected by three needs: need for power, achievement and affiliation [McClelland, 1961].
According to McClelland's Achievement Theory (1961), feedback is one of the key factors, which defines achievement-oriented performance [Sach, 2013]. Thus, an employee will expect unambiguous feedback from the source that should be valuable for a feedback recipient. During achievement-oriented performance employees perceive their own responsibility (internal attribution) for the results of their work; if an employee perceives that the goal is achievable through his/her performance (internal attribution), an employee is more motivated to achieve the goal. As McClelland noted, an "individual’s need for power" can be demonstrated "through actions that would enhance or preserve a person’s reputation"[Sach, 2013]. This statement can be interpreted that individuals with a high need for power will be more motivated to perform better to enhance their reputation, therefore feedback is a key factor that determines the gap between the actual level and required level for an achievement.
Also, McClelland reports that individuals with "high need for affiliation", which is defined as a need of individuals to "belong" to and feel included in a social group, work harder to win others' approval, as a consequence, they may even pursue "high need for achievement" if their performance promotes being accepted by the other fellow coworkers. For individuals with "high need for affiliation" that can be obtained through work achievements, feedback from their team members is very crucial as it serves as confirmation of having an affiliation with the team. Sach also reports that senior or experienced software engineers have a higher need for affiliation rather than new software engineers [Sach, 2013].

3.1.4 Attribution theory

Attribution theory was introduced by Fritz Heider (1958), explains how individuals perceive the information and events that they have experienced. According to the Attribution theory, internal attributions refer to the personal/internal factors, such as personality, ability, motivation, etc., external attributions refer to the environmental factors of the individual, that are outside of the individual's control, e.g., luck, organizational rules, etc. [Inamori & Analoui, 2010].
The attribution theory helps to interpret the individuals' perception of the received feedback, and consequently, predict the impact on individuals from the received feedback.
In aforementioned theories internal attribution was indicated as a factor that has a strong impact on the personal and work outcomes, that will be discussed below:
In Job characteristics theory, the psychological state 'Responsibility of the outcomes' can be interpreted as the internal attribution, and this psychological state impacts on the personal and work outcomes, so that it can improve work motivation, work quality, satisfaction with the work, and decrease absenteeism.
In Goal-setting theory, one of the incentives for goal-setting - individuals' participation in the

setting of goals, can be indicated as the internal attribution, which according to Locke's theory increases motivation and commitment to a goal.
In Achievement theory, individuals with "high need for achievement" should perceive their own responsibility for the results of their work in order to be motivated to achieve their goals.

However, according to “Self-serving attribution” theory, which is derived from Attribution theory, in order to maintain their self-esteem, individuals attribute their success to internal factors and attribute their failure to external factors. [Inamori & Analoui, 2010]

3.1.5 Pygmalion effect

A leader, by holding positive expectations towards his/her employees, and inspiring them with his/her behavior, can improve self-efficacy of his/her employees which in turn contributes to the improvement of employees' performance - this effect is called the Pygmalion effect. Pygmalion effect is also called "self-fulfilling prophecy", as 'self-efficacy can be manipulated', and that in its turn will boost performance [Eden, 1992] [Mitchell, 2003]. Model of the Pygmalion effect presented by Sutton and Woodman suggests that the Pygmalion effect occurs when a perceiver's expectations lead to perceiver's special behavior toward the target, which influences the target's self-expectations, which in turn affects the target's behavior. Also they recognized that subsequent target's behavior or performance has an impact on target's and perceiver self-expectation (shown by dotted line) [Sutton et al., 1989] [Inamori et al., 2010]

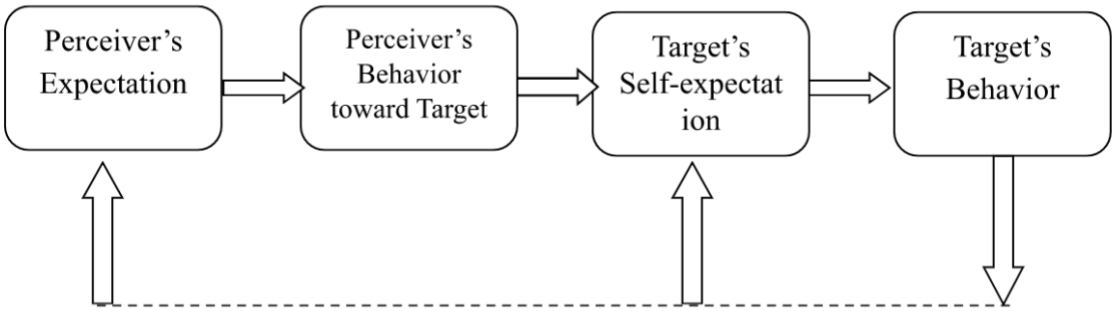

Figure 2: INTEGRATED MODEL OF THE PYGMALION EFFECT (Adapted from Sutton and Woodman)

3.1.6 Individuals external and internal aspects that are impacted by feedback characteristics

Based on interviews, diary studies and an experimental survey, Sach distinguished 10 feedback characteristics, which were reported by participants of the study as occurring in software engineering environments:

“Source: the person or machine sending the feedback, examples: colleague, manager, code compiler.
Goal: the intention of the feedback, examples: reduce stress, remove tension, encourage.
Medium: how the feedback is communicated, examples: verbally, email, phone, body language.
Direction: who the feedback is going to/from, examples: from one person to me, from multiple people to me, from me to one person.
Instigation: the prompt to provide feedback, examples: end of a project, arising issues, annual review. (must be together with goal)
Setting: the contextual environment when receiving feedback, examples: casual chat, individual meeting, team meeting. (Medium)
Timeliness: the time difference between the instigation and the sending of feedback, examples: instant, minutes, months.
Content: the polarity and topic of the feedback, examples: positive feedback about task performance, negative feedback about attitude, negative feedback about progress.
Preparation: what needs to be done prior to sending the feedback, examples: producing a document, compiling data.
Recipient: the person/s receiving the feedback, examples: team, individual, division.”

Additionally, to above-mentioned characteristics, we suggest two more characteristics as they also impact on the individual's perception of the feedback.
Quantity of Recipients: was feedback provided to a whole team or an individual;
Quantity of Feedback Sources: was feedback provided by one person or multiple people.

Feedback characteristics that change impact of feedback: source, medium, timeliness, recipient, setting. As for setting, it is recommended to "praise publicly; correct privately" that positively affect the impact on the "developers' trust and motivation"

Depending on the values of feedback characteristics, the impact of received feedback also changed.
Our model Perception and Further Impact of Positive Feedback was adapted from Sach's model of Impact of Received Feedback in Software Engineering.

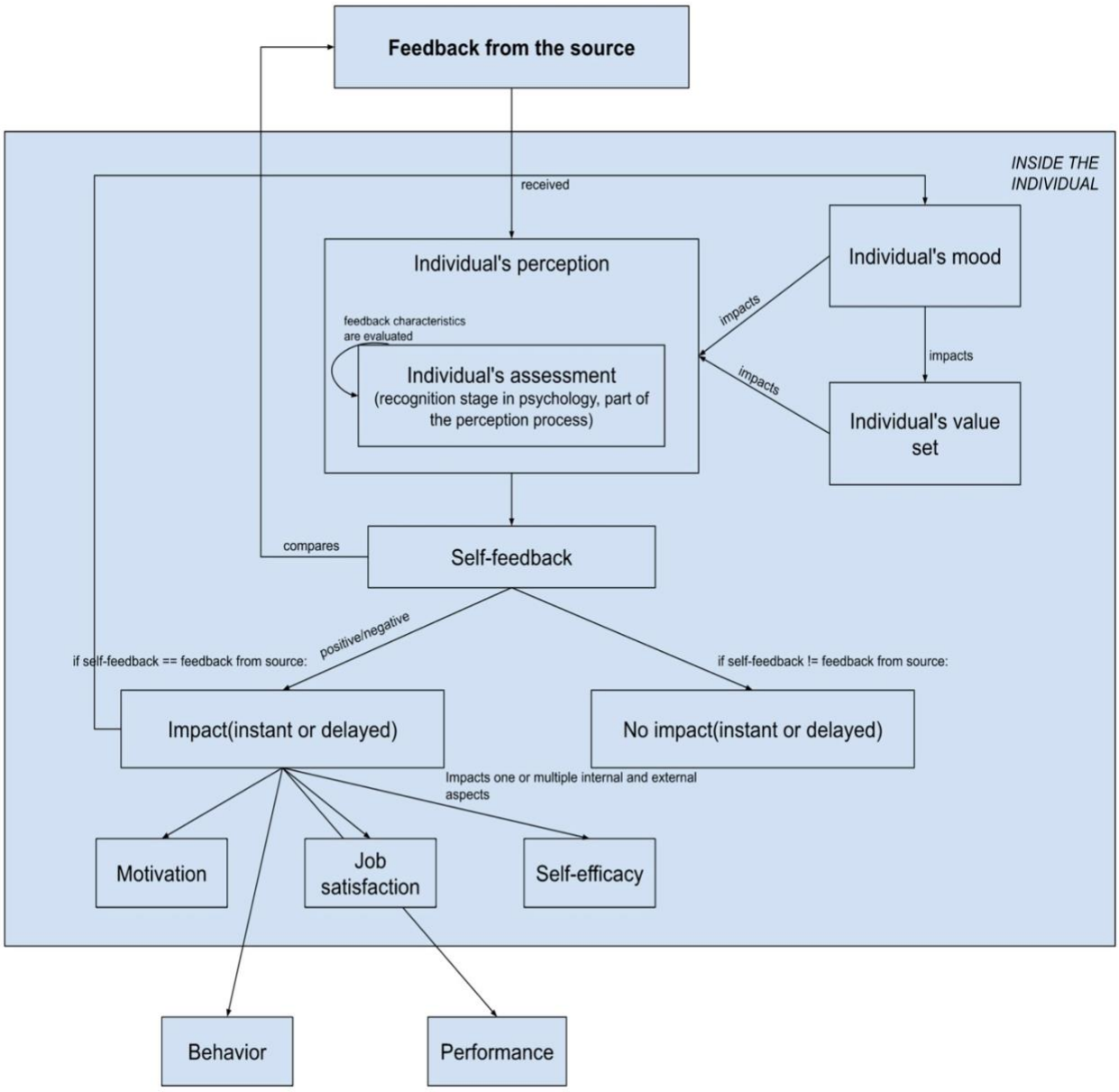

Figure 3: PERCEPTION AND IMPACT OF POSITIVE FEEDBACK IN SWE (Adapted from R. Sach's Model)

When positive feedback is received, the feedback recipient starts to perceive feedback. In contrast to the Sach's model, where the perception and assessment are sequentially separated, in our model we have placed assessment as part of the perception. As in psychology, the recognition stage[1], which is part of the perception process, has a similar description to the Sach's assessment stage. During the recognition stage, an individual categorizes, interprets and gives a meaning of the received information [Kenyon, & Sen, 2015]. Consequently, during this stage, received feedback is categorized into feedback characteristics, which are evaluated

[1] In various literature, this stage is also called as an interpretation and evaluation stages.

according to the individual's value set. Mood in its turn also impacts on an individual's value set, the perception process of the feedback and self-feedback. Individual's value set has a very strong impact on self-feedback.

Individuals assess not only the received feedback, but also their performance, this process is called "self-evaluation" [Weiner et al., 2003], in our research we will use a term "self-feedback" instead. The concept of self-evaluation helps to predict job satisfaction, motivation, performance [Weiner et al.,2003]. During literature review, we found some controversy as most studies indicate only positive impact on individuals from receiving positive feedback, however, there are many studies that in contrast to the others accentuate on the negative impact or no impact on individuals from receiving positive feedback. Therefore, we decided to add self-feedback as it helps not only to explain the atypical impact of positive feedback on people, but also to predict what impact it will have on individuals.

After the perception process, feedback recipient generates his opinion on received feedback, which is compared to the self-feedback. If received positive feedback, after the perception process, matches with the self-feedback, feedback is most likely to positively impact on the feedback recipient, but in case if positive feedback was received but an individual's self-feedback on performance was negative or lower than the received feedback, feedback is most likely to negatively impact the feedback recipient. Depending on the feedback characteristics, mood, and value set, positive feedback may have no impact on individuals. Moreover, if self-feedback matches with the received feedback, it might also have no impact on individuals. Intuitively, if an individual receives frequently positive feedback for a good performance, and if an individual doesn't chase career goals, after some time an individual might less appreciate it, as the result positive feedback might have no impact on that individual. Likewise, received feedback that doesn't match with the self-feedback might also have no impact on individuals. For example, when an individual receives positive feedback when negative feedback was expected (self-feedback), in most cases that will have a negative impact on individual's performance, as an individual might reduce performance standards, but it might also have no impact on individuals with combination of feedback characteristics that lowers the importance of the received feedback.

In case if any changes in feedback recipient's mood or value set appear, self-feedback and assessed feedback might be changed as well and if so, it will be again compared with the received feedback, and if they match, the impact from feedback occurs with delay.

Many research papers have studied how feedback affects an individual, and how an individual, after receiving feedback, affects the entire team. However, feedback given to the entire team (quantity of recipients) can impact on each team member differently, and in this case self-feedback can serve as an indicator of how the feedback will impact on each team member. For example, a team member that has invested the most of his time and effort on the project that was praised, will be much more positively affected by positive feedback compared to a team member who hasn't contributed as much to the success of the project.

Our proposed feedback characteristic – quantity of feedback sources, can also help determine the impact level of the received feedback, e.g., positive feedback provided by more than one feedback source has a much greater impact on an individual than by one source. We also hypothesize that feedback received from multiple sources can even impact on self-feedback, but it needs in further research.

If the impact occurs, received feedback can impact on one or combination of internal and/or external human aspects, which are discussed in the following parts:

Mood[2]

In many studies, it was found that the mood of individuals impacts on their performance [Ortu et al., 2015], [Ortu et al., 2018], [Mantyla et al., 2016], [Graziotin, 2018]. Also, Kaufman concluded that a positive mood unconditionally enhances creative performance. In recent studies, Graziotin et al. have reported that higher productivity and performance are most frequently appeared as aspects that benefit from happiness in the software development process, as it was explained that happy developers are more inspired to work on "undesired tasks" [Graziotin, 2018]. Happy developers are more focused, better in problem-solving, better mental energy and are better learners [Graziotin, 2018]. A good mood has a beneficial impact on specific programming activities, such as debugging. Additionally, Ortu et al. found a link between a developer's emotional state and the amount of time it takes to fix a software issue.

Sach stated that perception of feedback by feedback recipients and their reaction to it depends on their mood, moreover, feedback in its turn impacts on feedback recipient's mood. Therefore, Sach recommends feedback providers - feedback source, to consider the potential impact on the feedback recipient's mood when providing feedback, as the feedback recipient may misinterpret the actual message under the influence of emotions.

Feedback can be distorted by the current mood of the feedback recipient, however, feedback can also have a strong impact on software engineers, it can also impact an individual's mood, such as motivation, job satisfaction, self-efficacy which as a result will affect productivity and performance of an individual.

It is highly recommended to keep a positive and peaceful atmosphere in a team, as happy developers are more motivated to work harder.

Behavior.

Behavior plays an important role in the workplace, as an employee's behavior impacts on his/her performance and therefore, on an organization's productivity. Lenberg et al. stated that mainly SE research is concentrated on the technical aspects of the work, however some human behaviors as resistance to change [Oreg, 2003] could have significant negative impact, and need to be changed for software process improvement rather than changing a process or tool [Unterkalmsteiner et al., 2012.], [Lenberg et al., 2015]. Besides that, there are positive behaviors that can positively affect the productivity of an organization: a leader, by holding positive expectations towards his/her employees, and inspiring them with his/her behavior, can improve self-efficacy of his/her employees which in turn contributes to the improvement of employees' performance. This psychological phenomenon is called the Pygmalion effect, or Rosenthal effect [Mitchell, 2003] [Eden, 1992]. Eden et al. reported that managers that expect more from their employees lead them to greater accomplishment.

Manager, by his behavior, can demonstrate his/her expectation in relation to his employees. Manager can translate his expectation towards his employee through feedback on his employee's performance. Positive expectations can be expressed by feedback that emphasizes the best qualities of the employee and inspires him that he is capable of more. If the employee successfully perceives the feedback, the feedback will positively impact on the employee's self-expectation, as a result, it enhances employee's performance.

[2] R. Sach presented individuals' mood/state of mind/emotions as factors influencing the perception of the feedback by individuals; individual's feelings - as an internal aspect that could be affected by the feedback. However, in other SWE research papers feedback is indicated as a factor that impacts on the feedback recipient's mood. For the avoidance of doubt, in our study 'mood' is used as a unifying representative of mood, state of mind, emotions, feelings

Feedback can impact on an employee's behavior, as positive feedback will "reinforce and confirm" the desired behavior, without feedback positive behavior will more likely "be discontinued", after receiving negative feedback an employee will more likely change his/her behavior [Sach, 2013].

Motivation.

Motivation has been reported by many researchers and practitioners as having a positive impact on an individual's performance, productivity [Boehm, 1981], [Beecham et al., 2008], software quality [McConnell, 1998], and the overall success of the project[França et al., 2011], [Sach et al., 2011]. According to Software Engineering Institute data, 70% of the costs of SE projects are allocated on human resources [França et al., 2011]. Moreover, it was reported that demotivated software engineers can cost from 40% to 60% of a project budget [Sach et al., 2012], [Abdel-Hamid, 1989].

Based on theories of motivation: Job Characteristics Theory, Achievement Theory, Goal Setting Theory, feedback is a factor that has an impact on motivation, as feedback provides feedback recipients with "the knowledge of the result of their actions"[Sach, 2013]. According to his findings, Sach reports that in software engineering the impact of the feedback depends on "the values of the feedback's characteristics". Also, according to Attribution theory, output feedback is linked to performance [Kohli & Jaworski, 1994], also we know from previous research works that motivation affects an individual's performance and productivity [Boehm, 1981], [Beecham et al., 2008]. An individual's motivation to work increases when an individual perceives the outcome/output feedback as being caused by internal attributions, and correspondingly, an individual is demotivated to work when he/she perceives the outcome/output feedback is outside of his/her control, caused by external attributes. Internal attributions in Attribution theory refers to "the personal factors such as personality, ability and motivation", and external attributions refers to the individual's "environmental factors such as organizational rules, luck and natural environment"[Inamori & Analoui, 2010].

In contradiction to the above-mentioned research works, Sach's findings demonstrate that software engineers do not link motivation to productivity. According to his findings, positive feedback has an impact on motivation in 64% of scenarios, but productivity was associated with motivation in only 20% of positive feedback cases. While, in scenarios of negative feedback, productivity was associated with motivation more than 2 times more (percentage wise) often than in scenarios of positive feedback. This contradiction can be explained in the following scenario-examples, that are based on the Impact and Perception of Positive Feedback Model:

Scenario 1: when positive feedback is received, a recipient starts to perceive positive feedback and categorize it into feedback characteristics, which are evaluated according to the individual's value set. Employee has high measured needs for growth, and he/she perceives that career growth potential is dependent on his own abilities (internal attributions), career wise it's more significant for the recipient to receive positive feedback from someone who can advance their career, e.g., manager. Thus, positive feedback from someone who is not holding a position to impact their career could theoretically improve mood, behavior, attitude towards and boost motivation, but it wouldn't affect the feedback recipients' productivity or performance.

Scenario 2: recipient assesses positive feedback according to the feedback characteristics, his/her own set of values, current mood. If received feedback matches with the recipient's self-feedback on his performance, therefore, positive feedback serves here as a confirmation that a

recipient is doing well with the current level of performance and productivity, therefore, it is more likely that a feedback recipient is motivated to keep working on the same level of performance and productivity.

Scenario 3: recipient of the feedback identifies feedback characteristics of the received feedback at the perception process, which influence assessments of the feedback validity and value. During assessment, feedback is assessed according to his/her own value set and earlier received feedback characteristics. If the recipient performs to his/her full potential, positive feedback is perceived as a reward, as it demonstrates that the recipient's performance on full potential is paying off. As a result, this further motivates the recipient to work at that level.

Motivation plays a significant role in the Pygmalion effect, as the motivational behavior of the leader positively impacts the performance of the subordinate, which as a result enhances the productivity and performance of the team and organization as a whole [Inamori et al., 2010].

Job Satisfaction.

Feedback was identified as "the most consistent predictor of job satisfaction"[Carayon et al., 2003. Chen reported that “jobs with the features of feedback, professionalism and autonomy can most easily increase the job satisfaction of IS personnel” [Chen, 2008]. According to the Hygiene Theory, job satisfaction is linked with achievement and recognition [Herzberg et al., 1959]. Feedback forms such as "act of notice, praise, or blame" were identified as recognition, which was reported as "the second strongest motivator factor"[ibid: Herzberg et al., 1959]. Also, findings from Sach's research confirm that software engineers perceived received feedback as recognition, as it showed them that their “work at the company is appreciated"[Sach, 2013].

Herzberg determined that "turnover of staff, attitude towards the company, mental health and interpersonal relationships" are linked to job satisfaction [ibid: Herzberg et al., 1959]:

Turnover of staff: dissatisfied employees will more likely quit their job. Moreover, Herzberg reported that some individuals instead of physically quitting their job, they are "psychologically quitting", as it was explained they put in less effort at their work, only what is required to not lose their job.

Attitude towards the company: employees that are satisfied with their jobs are more likely to have a positive attitude towards their company and vice versa.

Mental health: research studies in psychology showed that job satisfaction can be directly and indirectly correlated with mental health [ibid: Herzberg et al., 1959], [Lee et al., 2009], [Faragher et al., 2013]. However, good mental health is not necessarily linked to job satisfaction [ibid: Herzberg et al., 1959].

Interpersonal relationships: Herzberg found a little connection between job satisfaction/job dissatisfaction and interpersonal relationships, as he explained it was more likely because “the degree to which a person lets his feelings about his job spill over into the conduct of his interpersonal relationships is more a function of psychological dynamics as an individual than of anything else” [ibid: Herzberg et al., 1959].

Kim and Wright examined how performance feedback impacts work exhaustion of employees; their studies confirm that performance feedback from supervisors could decrease **their** employees’ work exhaustion and turnover intentions, "by increasing role clarity and perceived advancement opportunities"[Kim & Wright, 2007]. Sach's study that was based on software

engineers reports found that positive feedback typically led to a positive impact on the engineers' job satisfaction (89% of all occurrences), and negative feedback typically led to a change in their behavior [Sach et al., 2012].

Therefore, we can conclude that positive feedback will most of the time increase employee's job satisfaction, as Sach's findings demonstrated. When positive feedback is received, an individual's feedback characteristics, mood, internal/external attribution, self-feedback, and full potential can change the impact of received feedback, but not significantly, however, these factors might significantly change the impact of negative feedback.

Consequently, it is recommended to managers to provide team members with "flexibility to balance work and personal life", "to energize the team" through events, celebrations of team successes as this practice helps to "maintain a positive working environment", which in its turn increases team members' job satisfaction [Kalliamvakou et al., 2017].

Self-efficacy.

Originally, the self-efficacy concept was proposed by Bandura. Self-efficacy was defined as individuals' level or strength of self-belief in their own capability to accomplish a task and achieve a goal [Bandura, 1977, 1986, 1997]. By many theories, such as Locke's Goal-setting theory, Bandura's Social learning theory, self-efficacy has been recognized as a crucial predictor of performance results [Rasheed et al, 2015].

As studies have shown that self-efficacy has a strong impact on motivation and performance, that's why a significant number of studies has been devoted to understanding its causes and developing training methods that would increase self-efficacy.

Self-efficacy may motivate individuals to seek feedback and select more effective task strategies. As studies have shown, individuals with higher levels of self-efficacy tend to choose much more challenging tasks, put out more effort and stick with a task longer than individuals with lower levels of self-efficacy. Self-efficacy was also linked to the ability to stay more focused and less distracted.

According to Bandura, self-efficacy and goals are influencing feedback effects. Self-efficacy is extremely important when "negative summary feedback" is provided, since the individual's level of self-efficacy after receiving that feedback decides whether subsequent goals will be raised or dropped.

Many studies demonstrate that self-efficacy has a strong impact on motivation and performance.

When individuals notice that they are not provided with the reward, the levels of their self-efficacy and their goal drop, as a consequence, the level of their performance also drops. [Locke & Latham, 2002] We can assume that positive feedback can be also considered as reward, and therefore lack of positive feedback will decrease the level of self-efficacy and goal, which in its turn will negatively impact on performance.

One of the three methods of "building efficacy" is "persuasive communications", which reproduces Eden's original concept of the Galatea effect [Bandura, 1997], [White & Locke, 2000]. This method aims to increase self-efficacy of employees, by having managers convey belief in the ability of their subordinates to reach the goals through persuasive communications.

White and Locke noticed that it might appear that for individuals, providing positive feedback must come effortlessly, however, researchers consider "giving positive feedback as a skill, which should be trained" [ibid: White & Locke, 2000]. White and Locke reported that encouraging positive feedback is "likely to build self-efficacy", if feedback was perceived as credible by the feedback recipient [ibid: White & Locke, 2000]. Positive feedback can be perceived as arbitrary(not valid) by the feedback recipient if feedback wasn't consisting reasons or there was no context behind(instigation) the feedback, therefore, it is recommended to managers to provide positive feedback with information that will explain why they believe that their subordinates can attain goals(e.g., feedback recipient has enough experience, has been well trained, etc.) or by sharing the strategy or the method that can help them to achieve their desired goals[ibid: White & Locke, 2000]. Also, positive feedback which has an encouraging and inspiring impact on a feedback recipient, is beneficial in preventing employees' resentment or discouragement.

The second method of building self-efficacy is role modeling or finding someone with whom individuals can relate with, and consequently improve their performance, as in Goal-setting theory, where it was also recommended to managers provide feedback in such a way that feedback recipients could have compared their performance to desired "standard"[Locke, 1968]. Managers by using models with higher performance as a target level of performance, can help their subordinates to develop high expectations for themselves. This method can be used in both negative and positive feedback.

Self-efficacy has a positive impact on many aspects of the individuals. Individuals with higher self-efficacy establish considerably more ambitious goals for themselves and they are also more devoted to challenging tasks(goals) that are entrusted to them [Bandura, 1997]. Moreover, people with a high level of self-efficacy are likely to perceive "performance setbacks" more positively [Bandura, 1997].

The Galatea effect is a self-fulfilling prophecy, and it is also a subcategory of the Pygmalion effect. It occurs when an individual's high level of self-efficacy of themselves leads to high performance. Managers can also positively impact the self-efficacy of their subordinates, by providing them with inspiring positive feedback that will convince them that they can achieve their goals. An indirect behavior of the manager may raise the subordinates' high expectations for themselves [ibid: White & Locke, 2000]. In contrast to the Pygmalion effect, the focus in the Galatea effect is on individuals' own expectations and not on the expectations of the superior(manager). A Galatea effect is also possible without a Pygmalion effect, as employees' can raise their own expectations from themselves.

Galatea effect can occur in various circumstances:

1) as unexpected result of the Pygmalion effect, e.g., unintended inspiring behavior (can be expressed through feedback) of the leader in relation to subordinates, affects subordinates' expectations;
2) as expected, result through persuasive communication from leader to subordinate.
3) as a result of indirect expectancy communications by the leader through role modeling.

Some studies assert that the Galatea effect is more concentrated on building self-efficacy beliefs indirectly, e.g., through modeling, rather than directly manipulating expectations [ibid: White & Locke, 2000].

Subordinates might develop high expectations for themselves if they are provided with models of high levels of performance for them to follow.

Performance.

Sach in his model represents 'productivity' and 'code quality' by the 'performance' aspect. Productivity and code quality were reported by software engineers as aspects that could be affected by received feedback [Sach, 2013].

According to the Locke's Goal-setting theory, feedback plays a major role in performance, as it provides individuals with the understanding of how productive their performance was (knowledge of score), and also some specific forms of feedback, such as "praise and reproof", are recognized as performance-influencing factors [Locke, 1968].

Hackman and Oldham found that individuals with "high growth need strength" has stronger correlation between the "three psychological states" and the "outcome variables" rather than individuals with "low need for growth"; also, individuals with "high growth need strength" has stronger correlation between the "core job characteristics" and their "psychological states" rather than for low GNS individuals. Consequently, high GNS individuals will be better able to perceive "the psychological effects of an objectively enriched work" and will be more likely to respond positively to that experience. From these statements we can conclude that "employee growth need strength" can change the impact of received feedback on a recipient [Hackman & Oldham, 1976].

Recent studies have shown that positive emotions and politeness that are expressed in comments by contributors have a positive impact on software developers' productivity in respect of issue fixing time, performance, and attractiveness of a project for new potential contributors.

Locke identified a varying effect of praise or reproof on motivation. He found that praise was effective in improving overall performance, and that reproof appeared to improve the performance if feedback was provided in relation to a standard.

As it was mentioned earlier, feedback provides individuals with the knowledge of the effectiveness of their performance and influences the psychological state of ‘knowledge of actual results’, which affects personal and work outcomes.

In Goal setting theory forms of feedback, as praise and reproof, are also known as factors impacting performance.

3.2 Code review as a feedback mechanism that affected by human aspects

In the software lifecycle, code review is a usual practice targeted to improve the code quality. Basically, the applied patch is inspected and evaluated by team members. In the traditional code review, software artifacts are inspected independently, however, in contemporary code review, initiated patches are reviewed in synergy by all team members, which is provided by a transparent environment with an open access to reviewer's feedback [Thongtanunam et al., 2020].

Code review positively affects team members by stimulating them to "be less protective about their code", sharing with code insights within the team, and therefore code review cultivates knowledge sharing [Bacchelli, 2013]. Transparent environment of contemporary code review in addition to aforementioned beneficial influences, prompts an active and timely collaboration and creates a democratic environment in a team. [McDonald, 2013]. Also, participation in OSS projects as it was reported by McDonald motivates developers to be more “democratic” and “transparent”, which attracts more contributors and therefore, more reviews and feedback are expected, which as a result leads to better outcomes.

Code review is a feedback mechanism that, depending on the quality of the code, can unconsciously develop the reputation of the code author, build relationships between the code author and code reviewers.

Bosu et al. reported that previous interaction, relationship with the code author, as well as the code author's reputation affect the reviewer's decision of whether to accept a review request. From previous interaction with the code author, a reviewer might form an opinion about a code author's level of competency [Bosu et al., 2017]. Intuitively, the code review from a trustworthy author has a higher likelihood to take less time and effort. Consequently, a code reviewer can anticipate the amount of time and effort that might be spent on a code review by having a prior interaction with the code author, or by knowing the code author's reputation. To utilize time on code reviews efficiently, a reviewer might choose to accept or ignore a code review request.

In its turn, high quality code creates positive perceptions about personal characteristics, helps build relationships, and encourages future collaborations [Bosu et al., 2017].

Tsay et al. reported that the developers' evaluation decision to accept a pull request is based not only on the technical value but also on the social relations between the submitter and evaluator. Also, Dabbish et al. pointed out that previous contributions of a potential new team member were used as a measure of the contributor's trustworthiness to a contribution. Bosu et al. observed that gaining a good reputation is one of the key motivations of OSS developers, as a developer's reputation has an impact on code review outcome, as studies have shown that developers without "an established reputation" wait "2 to 19 times longer" than the core developers to get feedback, and this delay negatively effects on motivation of developers and therefore on the overall result of the project [Bosu et al., 2014].

A recent study by Thongtanunam et al. hypothesized that a reviewer's evaluative decision to accept a proposed patch may be subconsciously biased due to human factors, which may affect code reviewer's "objective evaluation". To capture the visible information about a proposed patch, Thongtanunam et al. [Thongtanunam et al., 2020] has proposed three factors: 1) prior feedback, 2) status, and 3) relationship.

Prior Feedback. In a transparent environment reviewers’ feedback, as comments and votes, is visible to all team members. Due to human factors of reviewers, there is a tendency for reviewers to get influenced by the prior feedback of other co-viewers. Recent study by Thongtanunam et al. reports that prior reviewers' feedback is significantly associated with the evaluation decision of a reviewer. Subsequently, a proposed patch with much prior positive feedback is more likely to receive positive feedback again or to be accepted by subsequent reviewers.

Status. Also, studies have shown that not everyone's feedback draws attention during a code review. Moreover, it was found that non-core members should wait for feedback to their patch a few times longer than core members. Those cognitive biases are triggered by social hierarchy,

where implicit or explicit ranking of group members along a valued dimension influences behavior of others. Moreover, it is more likely that a patch from an author with the status, e.g., core members, will be accepted rather than rejected. In OSS projects, code reviewers are encouraged to rapidly provide feedback on code review requests, so that potential developers are motivated to make further contributions [Bosu et al., 2014].

Relationship. Code reviewers may build relationships due to regular interactions regarding code review. The long-term relationship may form a trust between them. Therefore, newcomers often face difficulties, their feedback is not considered as important during a code review process, as well as they receive feedback with a longer delay than the core team members. Research study by Thongtanunam et al. confirms that the ratio of the frequency of cooperation between a patch author and a reviewer, is related to the likelihood of receiving positive feedback or acceptance of a patch by a reviewer, and that has a small correlation with error proneness [Thongtanunam et al., 2020].

Sentiments in the code review

Feedback in a code review in addition to technical information, can contain positive and negative sentiments that affect the feedback recipient. Feedback can be considered as positive feedback if the code review consists of only positive sentiments, and if no negative sentiments were detected [Jongeling, 2015]. Sentiments affect the feedback recipient's mood. As emotions and motivation are linked to each other, a developer with an improved mood is more motivated to work harder which might lead to a raise in productivity. Therefore, time spent to complete code review and its outcome depends on the type of sentiments used in feedback for code review [Asri, 2019]. In OSS projects, negative sentiments in code review affect developer's leaving [Kaur, 2021], as "quit over mistreatment" [Zijie, 2021], [Graziotin, 2014]. A research study made by Khan et al. reports that a good mood has a positive effect on some programming tasks, such as debugging. Moreover, Ortu et al. has recognized a correlation between developer's emotional state and software issue's fixing time. Also, their findings have shown that happy developers were frequently expressing positive sentiments in their feedback, and it turned out that the issue fixing time was shorter, feedback with negative sentiments had a longer issue fixing time.

That's why project leaders were advised to keep abreast of the mood of the team members in order to solve any problems that can negatively affect productivity as well as to boost factors that have a positive impact on productivity [Ortu et al., 2015].

3.3 Interview analysis

First interview was conducted with the Software Engineer (SE) consultant, the interviewee has identified that requirements are not always valuable for system testers, and in the end system testers should verify and validate the quality of a software product by using requirements as a source. Poor requirements engineering creates a hurdle and sometimes even havoc in a development process, as it directs the whole project, towards the failure state. Such a failure in the requirements engineering may cause a bad quality product or a surplus amount in costs, time, etc. Therefore, the consultant was motivating requirements engineers to cooperate with

system testers and timely collect feedback from testers regarding the requirements, to minimize and avoid poor requirements.

Second interview was held with the Requirements Engineering (RE) & Strategy consultant, he emphasized the importance of collecting feedback regarding the requirements from divergent team members that are performing different types of roles in a project (as divergent developers (software-, hardware-, function-), test manager, release manager, etc.) to create wikis for requirements management process. Also, interviewee has provided an example for positive feedback from a tester on requirement descriptions. Good requirements require diligent work, to correspond to the good requirements criteria, so the tester would be able to verify and accept them.

Third interview, that was conducted with the Architect & SE consultant, was having some interesting examples about positive feedback in requirements engineering. Firstly, an interviewee has noticed how common it is in requirements engineering to give feedback only when the requirements are poorly written but not when they are good. Therefore, in real life requirements engineers perceive by no reaction or no negative feedback on their requirements from testers as positive feedback. Another example was when a project manager for a project with high quality requirements and system provides extra time for his team as indirect positive feedback from project manager towards his team as he expresses belief in them by not pressuring them with deadlines and standing for them in meetings. So, trust can be considered as an indicator of positive feedback, and it also motivates a team. Also, the interviewee has noticed that it's not easy to convince a whole team to integrate some new practices in their daily work routine, as example he noticed best practices for impact analysis on proposed change-request, but if the key people from the team would be convinced to use new practices, so with the reward and encouragement, as a positive feedback for work improvements, it would motivate the rest of the team to integrate new tools and new practices in their work routine.

Fourth interview was with the RE and SE consultant. Interviewee has suggested to conduct a "lessons learned" round during RE meetings as in practice this method is particularly used by software engineers and architects. Also, interviewee has noticed the lack of feedback in the working routine of requirements engineers compared to software developers and software architectures, so he has proposed a Ticket Approval flag feature that will serve as feedback for a requirement engineer from a tester. He has noticed that there are easily configurable tools in a market for implementing Ticket Approval or other feedback functions, however, it's hard to assimilate new functions into a daily workflow.

Fifth interview was performed with the Product Owner. He as other interviewees has also mentioned the lack of the positive feedback in the industry. Even during the retrospective meetings where not only problems and negative sides should be covered but also the strong sides and parts that were successfully implemented. Strong and positive aspects are slightly mentioned, as the attention is mainly focused on the process improvements and making problems in the process more visible.

Moreover, an interviewee has encouraged to ask customers for feedback for a specific function that was implemented by their request. Especially, if the function has fulfilled customers' needs or even exceeded their expectations, customers will experience positive emotions for the enhancement of their experience, consequently feedback would be also positive. Therefore, this practice helps to create direct customer relationships, emotional bonds with customers and earn their trust and loyalty towards the product and company.

Sixth interview was given by a Business Analyst. He has also suggested a small function as "Thumbs Up" button for a requirements approval, interviewee considers this function as fast positive feedback for a well-written requirement. This method of providing positive feedback doesn't take time as it just requires "push a button". Alternatively, for other purposes, as discussion feedback in details, a special meeting can be arranged.

Additionally, interviewee, as other participants, has recommended regularly conduct positive feedback within the team meetings, and mentioned a German proverb, which says „if you haven't been criticized then it's already a praise", which demonstrates the habitual perception of complete lack of feedback as positive feedback. Also, the regularity of the feedback round may depend on the team size and the culture of communication. In the literature review it was found controversies regarding the team size as some huge enterprises as Microsoft are keeping team size under certain limits between 2 to 12 people [Demirors et al., 1997], as well as [Cito et al., 2018] recommends keeping a team size small and specifies it as one of the factors that is effective in devOps feedback loops.

Seventh interview was conducted with Project Leader and SW developer. Like the other interviewees, he also thinks that praise is rare during the development process as meetings are concentrated on solving issues and rarely on the positive aspects, because the main goal of the team is to finish the project on time.

The interviewee believes that positive feedback mainly impacts on human aspects: individual and collective. Impact on individuals: team members feel happier, more appreciated, have a strong sense of self-efficacy in a team, and are more motivated to work harder, but this impact has a short-term nature. Impact on a team: positive feedback creates a positive and friendly atmosphere in a team, as the communication within a team will have an appreciative manner. While the negative feedback will not only bring a negative atmosphere in a team but also team members would be afraid to raise their voice and express their opinion, as they may be afraid to receive negative feedback again.

Moreover, interviewee has hypothesized that positive feedback doesn't impact on productivity, which contradicts with the collected observations from a literature review, and that motivates to form a hypothesis that the lack of positive feedback in software engineering can be due to people not believing in its effectiveness for the productivity of the team and for the organization as a whole, as well as they think that the effect of the positive feedback is too short, so it doesn't motivate a team to spend that much time and concentration on providing positive feedback during their workflow.

Eighth interview was performed with the Project Owner. Firstly, he remarked that his team has integrated a feedback culture. Positive feedback is frequently performed in cases when it is deserved. He believes that this approach has cultivated a friendly and appreciative climate within a team, and it motivated each team member to speak up openly and directly. Moreover, interviewee has described a scenario where a requirements engineer has adjusted a template and received positive feedback, so positive feedback in that case has served as an approval for a change and confirmation that RE is "on a right track". In this scenario RE has reused a sketch inside the requirement ticket, which was already recognizable by developers and has formed positive feedback. Also, here interviewee has mentioned the importance of convincing critical person, the feedback of the critical person has impacted on the final feedback of the rest of the team, as interviewee has claimed: "by convincing one key person - you can convince them all", which also confirm statement by another interviewee.

According to the observations of Project Owner, some software developers are not communicative, so they may give positive feedback for e.g., agree on something, that in fact they didn't want to agree on, but it is visible from their behavior, as face expression and attitude that they are not happy about it, so this feedback can be interpreted as indirect negative feedback. Therefore, an interviewee has recommended to get to know each team member and their behavioral patterns, to identify their veritable feedback and to recognize their pain point when applicable.

The results from the interviews provide the practical evidence to support findings from literature review as well as to find controversies between analysis from literature review and interview analysis that motivate to form new hypotheses regarding the positive feedback.

Table 1: Interview analysis results in terms of positive feedback in software engineering

Ideas, suggestions, problems \ Interview No..	1	2	3	4	5	6	7	8
Lack of positive feedback in RE			✓	✓	✓	✓	✓	
Best practices in RE to receive positive feedback	✓	✓	✓					✓
Lack of positive feedback during retrospective as main attention is on problems					✓	✓	✓	
Recommend using positive feedback mechanisms				✓				
"Convince key people" by receiving PF from them			✓					✓
PF positively impacts individuals and a team: let team members raise their voice, feel self-efficacy etc.							✓	✓
Fast positive feedback feature as Approval ticket				✓		✓		

Contradiction with SLR						✓	✓	

The SLR with the interview analysis has motivated to support a hypothesis regarding the value of the initial positive feedback. Both interviewees have confirmed the importance of convincing key people from the team. In one of the cases, positive feedback from the key people has influenced the rest of the team to provide positive feedback as well. In the second case, when key people were convinced to integrate the new tool, positive feedback was provided as a mechanism to motivate others to also use this tool. Moreover, in the SLR regarding the code review on the transparent environment was found that prior positive feedback from reviewers (as positive comments and votes) has impacted on the evaluation decision of other reviewers. Also, it was found that the frequency of collaboration between the patch author and the reviewer has a positive impact on the likelihood of providing positive feedback and voting for patch acceptance. Consequently, metrics to provide positive feedback (whether prior feedback was provided, relationship between feedback recipient and provider, status of the feedback recipient and provider) have a strong impact on code review practices and their effectiveness. Future work should also focus on connecting prior positive feedback with the code review within a team and it should be checked whether metrics that were used for OSS code review were relevant for code review within a team, as well as the effect of prior positive feedback on the other decision-making processes during the development cycle.

3.4 Metrics, best practices to provide positive feedback in SDLC

In software engineering, quality is the crucial factor that defines the satisfaction of the customer/user with the software product, as well as the satisfaction of the manager and coworkers with the accomplished performance of an individual, which will be expressed with the positive feedback. Quality is also a determinant of a software project's success, especially, in a competitive market.

Each phase of the software development life cycle (SDLC) impacts on the quality of the software. Software development life cycle aims to develop high quality software that meets or even surpasses the customer's expectations, complies with the project deadline and expected costs. The development process starts with converting user needs into software requirements, creating a software design based on those requirements, implementing and testing the software. Hence, to respond to the RQ1: “What are metrics to provide positive feedback in software engineering?” we decided to determine how we can reach high quality of software artifacts of each phase of the SDLC and assess them with quality factors and metrics and by following best practices at each software development phase, which as the result will help to achieve the overall success of the software project.

3.4.1 Requirement analysis.

All software requirements are identified during software requirements analysis. This phase is very crucial in the SDLC, as during this stage software quality assurance baseline and development risks are determined. The outcome of this stage is the Requirements Specification which is established by agreement of both parties, and this document is used further as the baseline for the design phase. The RS is a document that describes the software that should be developed, and thorough review of software requirements are conducted prior to the more detailed system design phases, that aims to minimize redesign risks and avoid software failures. The goal of the requirements analysis and specification is to provide analyzed requirements extracted from the customers' needs to the designers, so that they will be able to establish a software design that will be further used as an input for the implementation stage of the software.
During this stage the main factors that will affect the software quality are completeness, consistency, correctness, modification and traceability, which are described in more detail below.

Completeness
Completeness refers to situations when specification contains all the requirements, which are fully specified in the specification, and no information left aside. According to Boehm [Boehm, 1984], specification can be considered as complete when it meets following essential criteria:
1) no information is left undefined or "to be determined".
2) no undefined objects or entities.
3) no information is missing from the document.
First and second criteria represent an internal completeness, which suggest that the information already exists and is closed. Third criterium is a document's exterior completeness is the subject of the third characteristic, which guarantees that the specification has all the data needed for problem identification. To achieve full specification completion is almost impossible as analysts are not able to determine what is initially lacking from the specification if they don't even know what they are looking for in the first place.

Consistency
Consistency relates to situations when requirements in specification don't contradict each other. In practice, it commonly happens that in order to prevent inconsistency, one or more requirements might be eliminated, however, it might cause incompleteness, on the other hand, by adding one or more requirements we can achieve completeness of the specification but that might cause inconsistency in specification. Therefore, it is important to find a balance between these two quality factors.
Correctness
Correctness is frequently defined as the accomplishment of a certain business goal, as a new system aims to solve customer's business problems. Consistency and completeness are typically considered to be the two components of correctness, e.g., improvement of the consistency of the requirements might decrease completeness and consequently, that will negatively impact on correctness.
Traceability
Traceability relates to situations when each requirement might be traced back and forward to the design, source code, test cases that verify that the implementation was done correctly [ibid: Saavedra et al., 2013]. The standard IEEE 830:2009 unites both quality attributes: "Traceable"

and "Traced" in a single property called Traceable [IEEE Standard, 2009].

Modification

The SRS's structure and style should be in a way to allow for any requirements modifications to be made easily, completely, and consistently while keeping the SRS's structure and style. Thus, it is recommended to have a unique name for each requirement as well as each requirement should be expressed independently from the other requirements, and each requirement shouldn't appear more than once in the Requirement Specification [ibid: Saavedra et al., 2013].

Techniques that are used to elicit requirements from the customers, are:

1. Personal Interviews
2. Questionnaires
3. Customer/market surveys
4. Observation
5. Demonstration of product prototypes or the product itself
6. Brainstorming

Prototype is the technique to let users understand the future software system and make sure that all requirements were included. Prototype in the technique on which users/customers provide their feedback on software requirements:

1. lack of features from the requirements specification.
2. modification of their requirements' implementation to better meet their demands.
3. wrong implementation of the requirements.
4. any additional requirements that were not previously included.

3.4.2 Design

After software requirements have been analyzed and modeled, software design is considered as the final step in the modeling process, as it prepares the ground for development (code generation and testing). During the design stage a model of the software is created, which is focused on describing software architecture, data structures, interfaces, components that are required for implementation.

Design is a multistep process in which representation of programs are built, which are focused on "the interrelationships of parts at the higher level and the logical operations involved at the lower levels".

Design model elements must be also traceable to the requirements model.

For evaluation of a good design, Pressman suggested following design concepts (software quality criteria) which "were achieved through the application of fundamental design principles, systematic methodology and review":

1. A design should exhibit an architecture that
    (1) has been created using recognizable architectural styles or patterns,
    (2) is composed of components that exhibit good design characteristics,
    (3) can be implemented in an evolutionary fashion, thereby facilitating implementation and testing.
2. A design should be modular; that is, the software should be logically partitioned into elements or subsystems.
3. A design should contain distinct representations of data, architecture, interfaces, and

components.
4. A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
5. A design should lead to components that exhibit independent functional characteristics.
6. A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
7. A design should be generated by information gathered through an analysis of the software requirements.
8. A design should be expressed with a notation in a way that clearly reflects its meaning.

[Pressman, 2005]

Software architecture is defined as "the overall structure of the software and the ways in which that structure provides conceptual integrity for a system" [Shaw & Garlan, 1995].
Software architecture determines the relations between software structural elements, the architectural styles and patterns that are used to fulfill the requirements, and in its turn the system constraints can affect architecture implementation. A software architecture is designed to accommodate the components.

As one of the best practices for achieving a high software quality at this stage, we can mention the technique that is used in Extreme Programming (XP) to enhance the quality of software at the design phase, which is called architectural spiking. This technique is used to reduce risks before the implementation begins and to increase the reliability of a user story's estimate. So, if a design problem is encountered, a design prototype for this dubious part of the design, which is called spiking solution in XP, is implemented to determine and test a potential architecture [Pressman, 2005].

Component level design takes place after the architectural design, where software components were defined. This design describes how data structures, algorithms, interface characteristics, and communication mechanisms assigned to each software component are elaborated to implement the software's desired performance. Component-level design allows inspecting the design for correctness and consistency with other design representations as data, architectural design, interface design, so that it defines whether interfaces, data structures and algorithms will function.

The quality of the component-level design can be evaluated by the cohesion and coupling of the components if a procedural design is available, since knowledge of the elements within a module is required to evaluate cohesion and coupling. Alternatively, cohesion and coupling evaluation can be done after the source code implementation.
Cohesion - indicates functional strength of the module, in other words, it demonstrates the extent to which the elements inside the module belong together. Ideally, a cohesive module should perform a single task, however, it is more essential and useful to have a single software component that fulfills numerous purposes.
Coupling - indicates the extent to which classes and components are connected to each other. It is important to minimize coupling, software that has simple communication between modules is simpler to understand and less likely to experience "ripple effects," which happen when faults occur in one place and spread throughout the system. To achieve minimum coupling of the

module's software architecture can be restructured without modifying a code. Refactoring is also used to solve cohesion and coupling problems, which is achieved by optimizing a component's design (or code) without affecting its functionality or behavior [Pressman, 2005], [Wang et al., 2018]. Refactoring also helps to improve the quality of the software system.
Functional independence is essential to effective design, which as the result impacts on the software quality. Functional independence can be achieved by high cohesion and low coupling of components.
Component qualification makes sure that a component meets its specifications and suits into the architectural style specified for the system and corresponds to the quality characteristics, such as performance, reliability, and usability that are necessary for the software.

Design Pattern

Design patterns give proven solutions to well-known issues and promote reuse and spare time and effort of programmers from "reinventing the wheel" [Pressman, 2005], [Singh, 2016]. The use of design patterns is a good practice both in the development phase (more efficient and effective program design) and in the maintenance phase of the program (faster and more accurate program comprehension) and as a result it has a positive impact on software quality and productivity.
Design patterns help designer to determine:

1) whether the design pattern can be relevant for the current project,
2) whether the design pattern can be reused, so it will help to save time,
3) whether the pattern can serve as a guideline for the development of a similar but functionally or structurally different pattern. [Pressman, 2005]

Design patterns are stated to have positive effects at the design and implementation phases of software development life cycle [Prechelt et al., 2002], [Wedyan & Abufakher, 2020], such as:

1) supporting better design decisions,
2) enhancing communication both "among developers and developers with maintainers"(due to common design terminology),
3) strengthening or easing software maintainability and reusability,
4) helping to satisfy non-functional requirements of the system,
5) saving money, time, and effort by reusing tried-and-true solutions.

Design patterns have a positive impact on software quality, as their usage improves maintainability, completeness, reusability, stability, understandability. [Singh, 2016]

Modularity

One of the most fundamental software design principles - "separation of concerns" is achieved by facilitating modularity, which is accomplished by separating information into components, known as modules, that are integrated to meet the needs of given requirements [Pressman, 2005].
Effective modularity can be achieved by having each module be cohesive in its functions and/or restricted in the content it represents, focusing solely on one well-defined aspect of the system; also each module should have low coupling with other modules, etc[Pressman, 2005].

Modular programming is the design technique that makes a program intellectually manageable. Modularity aims to reduce complexity by splitting up functionality into degrees of interdependence and independence and hiding the "complexity behind an abstraction and

interface"[Baldwin & Clark, 2000]. The reduction in complexity leads to an increase in the level of understandability, which consequently reduces the cost of software development. However, the raise in the number of modules will cause the raise in costs for module integration, thus it is important to find a tradeoff to avoid "under modularity" and "over modularity".
The modularity of the design promotes distributed code development, testing, reuse and rapid system design, integration, maintenance and evolution. Moreover, effective modularity makes software maintainable.
Many studies state that software modularity can enhance productivity and quality, which are shown in the Table:
Sun et al. analyzed previous research studies on the impact of modularity on key performance measures in R&D development: engineering productivity (engineering hours per normalized project), total product quality (includes conformance quality and 'product integrity', a high-level measure of product performance and consistency with overall company identity), time-to-market (normalized project), and costs [Sun et al., 2014], as shown in the adapted Table. As Sun et al. studies show that they couldn't find sufficient proof whether higher software modularity decrease time-to-market, therefore time-to-market will be excluded as a benefits variable from the Table:

Table 2: Benefits of modularity for benefits for embedded software development (Adapted from Sun et al., 2014)

Benefits:	**Literature reference:**
Productivity	[Lim, 1994], [Basili et al., 1996], [Frakes & Succi, 2001], [Morisio et al., 2002], [Selby, 2005]
Quality	[Lim, 1994], [Basili et al., 1996], [Prieto-Diaz, 1993], [Frakes & Succi, 2001], [Succi et al., 2001], [Morisio et al., 2002], [Baldassare et al., 2005], [Selby, 2005], [Zhang & Jarzabek, 2005], [Gupta et al., 2009], [Gupta et al., 2010], [Lopez & Niu, 2011]
Cost	[Lim, 1994], [Tomer et al., 2004], [Zhang & Jarzabek, 2005], [Lopez & Niu, 2011]

The result of the research study by Sun et al. claims that software modularity improves productivity, quality, and reduces costs of software development and their research has shown that higher modular software modularity reduces the number of defects [ibid: Sun et al., 2014]. Therefore, companies are strongly encouraged to adopt software modularity.

Modularizing a design makes it easier to plan development, define and deliver software increments, accommodate modifications, test and debug more quickly, and perform long-term maintenance with minimal negative effects.

3.4.3 Development

During the development phase, the software system is developed according to the earlier created requirement specification and design documentation. It should also correspond to the coding standards and software quality characteristics. To assess source code quality, the following quality attributes are used: size, complexity, readability.

Size metric: Lines of code (LOC)
LOC is one of the earliest software metrics that counts a program's actual lines of code, omitting blank lines and comments [Zhang, 2009]. It is a size metric of the final source code, rather than a size metric which may be applied during the specification or design phases [Stockman et al., 1990].
LOC is one of the popular metrics in software engineering, it is used: "

- as a predictor of development or maintenance effort.
- as a covariate for other metrics, “normalizing” them to the same code density,
- as a standard against which other metrics can be evaluated."[Rosenberg, 1997]

Program's maintainability can be determined by the size of the units, larger units logically are much harder to maintain than lower units, as their levels of testability and analyzability are low [Heitlager et al., 2007]. Unit size is measured by lines of code metric. Heitlager et al. reported a strong correlation between size in terms of LOC and cyclomatic complexity, however they recommended to use unit size in conjunction with complexity, as they together facilitate detection of large units with low complexity. Consequently, this practice helps to more accurately calculate the level of maintainability.
LOC can also be used to estimate effort involved in testing or maintaining the code, which is based on the straightforward concept: the more code requires more effort, and consequently causes more bugs [ibid: Rosenberg, 1997].
Rosenberg recommended to use LOC as a "covariate adjusting for size in using another metric", rather than as a quality predictor, although a high correlation between software metrics and LOC was observed. [ibid: Rosenberg, 1997]
Despite the popularity and long-term experience with the LOC metric, there are still a number of debates about it. Procedural programs are rarely measured only to determine their length, LOC is rather used to calculate various metrics [Michael et al., 2002]. As it was used to compute Defect Density, which was acknowledged as "a de facto standard measure of software quality"[Fenton & Pfleger, 2014].
Defect Density is calculated as:
Defect Density = Defect count/size of the release,
where size of the release is measured in terms of lines of code.
Zhang reports that a simple software measure, as LOC, can predict the quality of software [ibid: Zhang, 2009]. Moreover, LOC is considered as a measure from which productivity metrics can be calculated [Pressman, 2005].

Size metric (volume, vocabulary): Halstead Metrics
Halstead's measurements are based on program execution and its results, particularly on the inspection of the operators and operands from source code. Halstead measurements make it possible to evaluate the program's vocabulary, difficulty, bugs, and programming efforts for the

source code of any programming language. Some of the Halstead metrics are also used in the testing phase, which is discussed in more detail later in the next phase.
Halstead measures were designed to make programmers think of each program as a collection of operators and its operands. According to Halstead's statement, "A computer programme is an implementation of an algorithm that is believed to be a collection of tokens that can be classed as either operators or operands."[Halstead, 1977].
The following measures can be collected, by counting and determining operators and operands:
n1 = the number of distinct operators,
n2 = the number of distinct operands,
N1 = the total number of operators,
N2 = the total number of operands,
n1* = Number of potential operators.
n2* = Number of potential operands.

By using these primitive measures, Halstead derived formulas for the overall program length, the actual volume (required number of bits to specify a program), potential minimum volume for an algorithm, program level (a measure of software complexity), the language level, programming effort, and number of faults in the software [Pressman, 2005], which are in details listed below:
Vocabulary of the program, n:
Sum of the number of distinct operators and operands.

$$\eta = \eta_1 + \eta_2$$

Length(size) of the program, N:
Sum of the total number of the operators and operands in the program.

$$N = N_1 + N_2$$

Estimated Program length:

$$\check{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

Volume of the program, V:
This measure was described by Fitzsimmons and Love: "For each of the N elements of a program, log2 η bits must be specified to choose one of the operators or operands for that element. Thus, V measures the number of bits required to specify a program."[Fitzsimmons et al., 1978]
$V = Size * (\log_2 vocabulary) = N * \log_2 \eta$ measured in bits

Potential minimum Volume, V*:
Halstead defined it as the volume of "the most succinct form in which an algorithm could ever be expressed"[Halstead, 1977].
V^* - measured in bits,
$V^* = (2 + \eta_2 *) \times \log_2 \times (2 + \eta_2 *)$, where η_2 is the count of unique input and output parameters.

The following metrics are applied to testing and software quality testing, but they are derived from the metrics:
Program Level, L:
Program level is used to differentiate low-level language and high-level language. It

corresponds to the phenomenon that programs that are written in low-level language are typically longer than identical programs that are written in high-level language [Flater, 2018]. Program level is derived through the ratio of potential minimum volume to the volume of the actual program. However, it measures artifacts of the program, not the language [Flater, 2018].

$L = \frac{1}{\left(\frac{\eta_1}{2}\right) \times \left(\frac{N_2}{\eta_2}\right)}$ measured in bits

Estimated program level:

$\acute{L} = 2 * \frac{\eta_2}{(\eta_1)(N_2)}$

Difficulty of the program, D:

The level of the program decreases, and its complexity increases as the volume of its implementation increases. Therefore, using redundant operands or not using higher-level control constructs will increase the volume and complexity.

$$D = (\eta_1/2) * (N_2/\eta_2) = 1/L$$

Programming effort, E:

Halstead described this metric as "the total number of elementary mental discriminations E required to generate a given program." [Halstead, 1977]

Programming effort measures the amount of required effort for implementing or understanding the program, it is directly proportional to difficulty and volume.

$$E = \frac{V}{L} = D * V$$

Number of delivered bugs in the program, B:

The number of delivered bugs correlates with the overall complexity of the software.

Delivered bugs shouldn't be more than 2 in a file.

$B = \frac{E^{\frac{2}{3}}}{3000}$

Ming-Chang proposes following software quality factors that can be assessed with the Halstead's software metrics, which are demonstrated in Table:

Table 3: Software quality factors that can be assessed with the Halstead's software metrics [Ming-Chang, 2014]

Halstead's software metrics	**Software quality factors**
Program Length, N	Maintainability Modularity Performance Reliability

Volume, V	Complexity Maintainability Reliability Simplicity
Potential Volume, V*	Conciseness Efficiency
Program Level, L	Conciseness Simplicity
Program Effort, E	Clarity Complexity Maintainability Modifiability Modularity Number of Bugs Performance Reliability Simplicity Understandability
Number of Bugs, B	Maintainability Testability

One of the main disadvantages of the Halstead Volume metric, which is used to compute the Maintainability Index, is that it is difficult to compute even with defined operators and operands, as some programming languages require full syntactic and partial semantic analysis [Heitlager et al., 2007].

Complexity metric: Cyclomatic complexity
Source code complexity is measured in terms of cyclomatic complexity, which can be used as a quality metric. Cyclomatic complexity is a software metric, which calculates logical complexity of the program, by measuring the number of linearly independent paths within the source code [Michael et al., 2002]. It is used to estimate reliability, testability, and maintainability of a program. Cyclomatic complexity is measured by creating a control graph representing the entry points, exit points, decision points, and possible branches of the program being analyzed.
Complexity can be measured at class or method level. High cyclomatic complexity modules are more likely to be error prone than modules with lower cyclomatic complexity.
Complexity can be determined in one of these ways:
1. Cyclomatic complexity is the number of regions in the flow graph.
2. Cyclomatic complexity V(G) for a flow graph G can be defined as:
$V(G) = E - N + 2$
where E is the total number of flow graph edges,
N is the total number of flow graph nodes.
3. Cyclomatic complexity V(G) for a flow graph G can be defined as:
$V(G) = P + 1$
where P is the total number of predicate nodes contained in the flow graph G.

Cyclomatic complexity helps:
1) to reduce the coupling of code. High cyclomatic complexity indicates highly coupled code, which is hard to modify, maintain and test.

2) to increase readability of the code. The higher complexity number the more independent paths which results to more defects and unexpected outcome.
3) to ease testing. Complexity number is equal to the number of independent paths, which equals the required amount of test cases to cover different paths though the code.

Code readability: Source code comments
Source code comments are used to give description on how the program is intended to work and how a user should perform a task. Therefore, comments are used to improve usability and maintainability of the software, which are known as software quality characteristics.
Moreover, percentage of comment lines per module is considered as one of the metrics that is used to determine Maintainability Index (MI).
The following aspects should be kept constant to achieve best practices for writing source code comments: "

- the absolute information content of the comments (because this is what makes comments useful),
- the length of the comments in lines or words (because reading comments consumes time and concentration),
- the amount of repetition of information in the comments (because repetition may aid understanding or finding information or may make the reader unattentive),
- the understandability of the comments,
- the comments' potential for creating confusion (because comments can sometimes make a program harder to understand),
- the placement of the comments with respect to the program entities they refer to (because that determines how easily information can be found or related to the program)."

[Prechelt et al., 2002]

These metrics are extremely important because they make it easier to spot defects and anomalies and fix them at a lower cost than during the maintenance phase. If sufficient quality levels are achieved during development, it is not required to invest further time and effort in improving the quality of the software product.

3.4.4 Testing

Software Testing is determined as a process of verifying and validating that the implemented software meets its technical and business requirements and identifies important defects/errors/flaws that must be fixed. [Bentley, SUGI 30]. Software testing is used to test the software for the following software quality factors, such as reliability, usability, integrity, security, capability, efficiency, portability, maintainability, compatibility, etc. However, software testing teams can only measure quality, but not improve it.
During the System under test, all errors must be found and fixed unless it hasn't reached an acceptable quality level.
Software testing can also be categorized by software development lifecycle phase, but contrariwise:

1) Unit testing.
   This testing ensures that each unit (component, class, etc.) corresponds to the elements of the component-level design and behaves as intended. During unit testing, all control

paths of the component are checked to detect errors.

2) Integration testing.
   Components are combined into groups, which should correspond to the element of the architectural design, which is integrated and tested to find errors in the behavior between integrated components. As a result, integrated groups should form an entire software system.
3) System testing.
   checks conformity of the integrated system with the software requirements. System testing ensures that the software system behavior corresponds to the user's/customer's expectations.
4) Acceptance testing.
   User acceptance testing, which is also known as beta-testing, is performed by the users to determine whether the solution presented by the system works for them, so that they accept the system.

Software is tested from two different perspectives:
1) internal program logic is examined using white-box testing techniques,
2) requirements are examined using black-box testing techniques.

White-box testing:

White-box testing approach is a testing method that tests the structure and flow of the software under test but not its functionality. Test cases are designed according to the internal structure of the software.

In the software testing process, white-box testing can be applied at the unit, integration, and system levels, but it is typically done at the unit level.

Techniques used in white-box testing:

Code Coverage measures what percentage of a program's source code has been executed by a specific test suite. A program with high test coverage, which means that more of its code was executed during testing, more likely will have less undetected software bugs compared to a program with low test coverage.

Coverage criteria is defined as "the set of conditions and rules that impose a set of test requirements on a software test"[Dawood et al, 2017]. Multiple test coverage criteria help to achieve better coverage. Coverage criteria are used to estimate the quality of test cases. Code coverage can be expressed a percentage for:

Function coverage - each function of the program called at least once,

Statement coverage - each statement executed at least once,

Branch coverage - every branch of each control structure traversed,

Condition coverage - each boolean sub-expression evaluated both to true and false,

Multiple condition coverage - all possible combinations of conditions inside each decision covered.

In *mutation testing*, original code is modified in small ways, so that mutated(faulty) programs are created, which are called *'mutants'*. Test case should detect a mutant, e.g., one of the tests fails, which is called *'killing the mutant'*. Test suites are evaluated by the percentage of mutants killed. New tests can be additionally created to search for programming errors more effectively. Thereby, mutation testing helps to create effective tests, find flaws in the tests and code parts that have never been accessed while being executed. As it was mentioned earlier, mutation testing verifies the correctness of the software. The only drawback of this testing approach is

its computational expensiveness, due to the great number of created mutants, the compilation and execution time increases.

Black-box testing

In opposition to the white-box testing, black-box testing essentially tests the functionality of the software but not its structure. Test cases of black-box testing are derived from the external descriptions as requirements, specifications and design parameters, without considering implementation of the code. During black-box testing, the tester knows the inputs, outputs to related inputs, and specifications, but the tester doesn't know how the outputs are produced by the software. Thus, the outputs of the software are observed to determine its functionality. During testing, a variety of inputs are submitted to the test, and the results are checked against the specification to ensure they are accurate.

Black-box testing can be applied to every level of testing within the SDLC: unit, integration, system and acceptance levels.

Techniques used in Black-box testing:

In *Equivalence partitioning*, the input data is divided into equivalent partitions that are used to derive test cases, in such a way that each partition must be covered at least once. Each partition must have one test case in order to assess the program's behavior for that partition. This technique helps to find unnecessary test cases, consequently, lesser test cases should be developed, that as a result reduces time required for testing because of the small number of test cases.

Boundary-value analysis is the next step after Equivalence Partitioning, where test cases are chosen at the boundaries, minimum and maximum edges, of the equivalence classes. In this testing approach boundary values of valid and invalid partitions are tested. These boundaries are usually tested in test cases because they are typical sites for mistakes that lead to software faults.

When software doesn't function as intended, "debugging" is the process of identifying, analyzing, and fixing errors. Software quality can be improved by efficient testing and debugging, which requires less work and produces better results.

Halstead metrics for Testing

As it was mentioned earlier, some of the Halstead metrics are used for testing and as well as determining testing effort (testing metric):

With the use of the program volume V and program level L, programming effort E can be derived:

$L = \frac{1}{\left(\frac{\eta_1}{2}\right)\times\left(\frac{N_2}{\eta_2}\right)}$ measured in bits

$$E = \frac{V}{L} = D * V$$

Halstead described programming effort metrics, as “the total number of elementary mental

discriminations E required to generate a given program." [Halstead, 1977]. Programming effort measures the amount of required effort for implementing or understanding the program, it is directly proportional to difficulty and volume.

The percentage of testing effort to a module k, can be calculated with the following approach, where $\sum e(i)$ represents the total Halstead effort across all the system's modules and $e(k)$ represents the effort required for module k.

$$Percentage\ of\ testing\ effort\ (k) = \frac{e(k)}{\sum e(i)}$$

The testing metric, number of delivered bugs B, correlates with the overall complexity of the software.
Delivered bugs shouldn't be more than 2 in a file.

$$B = \frac{E^{\frac{2}{3}}}{3000}$$

Reliability refers to a product's or component's ability to continue to perform its intended function consistently over time under predetermined circumstances [Fitzpatrick, 2000]. In statistical testing, statistical methods are used to evaluate reliability of the software, it is concentrated on how defective programs can impact its operation [Chillarege, 1999].
Software reliability is measured in terms of the mean time between failures (MTBF), the mean time to repair (MTTR), the mean time to failure (MTTF), and the probability of failure. MTBF MTBF consists of mean time to failure and mean time to repair, where MTTF is the difference of time between two consecutive failures, and MTTR is the time required to fix the failure. Every time a bug is fixed, the MTBF will increase [Chillarege, 1999], and the higher the MTBF, the more reliable the system, and that serves as the release criteria to stop further software testing. This metric is used to enhance a product's quality.

Table 4: Test metrics

Test metric	Definition/ Purpose	Formula
Test coverage	examines efficiency of testing over time	% of code branches that have been executed during testing
Test Improvement (TI)	shows the relation between the number of weighted defects and the size of the product release. Purpose: deliver a high-quality product.	Test improvement TI = number of defects detected by the test team during / source
Test effectiveness (TE)	shows the relation between the number of weighted defects detected during testing and the total number of weighted defects in the product.	Test effectiveness = ((Defects removed in a phase) / (Defect injected + Defect escaped)) * 100

Test metric	Definition/ Purpose	Formula
	Purpose: deliver a high-quality product.	
Test efficiency (TE)	determines the efficiency of the testing. It finds the defects that were missed during the testing phase, but were found during user acceptance testing	Test efficiency = (defects identified during testing(Test defects) / defects identified during testing (Test defects) + post-testing defects) * 100; Test defects = Unit testing defects + Integration testing defects + System testing defects. Post-testing defects (User acceptance testing) = Defects or bugs found by the customer or end user during the acceptance testing.
Defect density	1) find the number of defects in software; 2) high risks areas; 3) readiness of software/component	number of defects/ sizes of the release
Mean time between failures (MTBF)	the average time between repairable failures, Purpose: deliver a high-quality product, serves as a release criterium	MTBF = MTTF + MTTR

3.4.5 Feedback mechanisms and best practices to receive positive feedback in DevOps

DevOps are well-known for the best practices to automate and integrate processes between Software Development and Operation teams. Communication between those two structures is a key factor that makes this collaboration effective. In research by [Cito, 2018], it was investigated how the runtime data that was gathered during a production, was transferred to software development as feedback, also here were identified factors that organizations face by facilitating feedback loops between operations and the software development life cycle, as organizational size, nature of business, presentation of feedback, and case-specific technical challenges.

Factors that are effective in devops feedback loops:

- Small organizations have better communication as they know each other. As teams in small organizations are mostly located within a building, the ad-hoc feedback can be easily applied.
- Nature of the business also plays a major role in feedback mechanisms. Companies with a strong IT infrastructure, such as cloud computing, have rapid delivery life cycles that beneficially affect feedback communication.
- The way operational data is presented to stakeholders plays a vital role in speed of the decision-making. Therefore, feedback should be provided with an interface that can support developers' decision-making.
- Feedback from operations should be technically well-equipped so that different stakeholders as developers and product managers would be able to provide their review on collected feedback.

Operations feedback should be viewed in the corresponding phase of development. So, the following runtime metrics are mapping to a phase in the development lifecycle with an option to present this metric to stakeholders.

1. System Metrics (CPU utilization, IOPS, memory consumption, process information) should be viewed in post-CI, canary deployment, and be presented to DevOps or performance engineers as reports, dashboards, for Non-functional and performance testing.

2. Application Metrics (exceptions, logs/events, usage) should be viewed in-development phase, and be presented to software developers as dashboard, IDE, for Informed refactoring.

3. Application System Metrics (method-level response time, load, garbage collection metrics) should be viewed in Post-deployment, Canary deployment, and be presented to product owner, DevOps or performance engineer, as alerts, reports, dashboards, for User behavior, Integration test in production.

Four phases where feedback from operation is valuable:

In-Development - application system metrics of the IDE, provides developers with a feeling about runtime specifics of their methods and identifies hotspots in context. Further, mapping of exceptions and application specific messages (logs) with their distribution at runtime provides an indication of the number of exceptions and events in production.

Post-CI - after CI, applications can request feedback, as performance testing, it can take longer to obtain, and thus is not justified to interrupt the software development flow. System metrics can serve as a baseline to compare the performance results.

Canary Deployment - is used to reduce the risk of a change in production, a canary release only rolls out changes to a subset of users. Application and system metrics are essential to detect deviations from the baseline. The primary task of operations feedback is to discover any issues and potentially roll-back or roll-forward.

Post-deployment - application and system metrics are observed. Moreover, different stakeholders are involved in the analysis that tend to be tied for long-term goals. In that stage, an operations engineer observes workload patterns, or a product owner plans upcoming releases and prioritizes features.

3.5 Positive feedback from users/customers

Customer/user feedback is a valuable resource to follow changing customer's tastes in the competitive market. The indicator of success of a software product is whether a product satisfies all needs and expectations of the user. A satisfied user will recommend a software product by providing a positive review, which will lead to the acquisition of new customers, respectively, a negative review will damage the business reputation, which intuitively will lead to the loss of potential customers.

Customer satisfaction is one of the most important aspects in any market since it can make the customers loyal and generate recurrent revenues. In more details customer satisfaction will be discussed in the next chapters.

3.5.1 Positive feedback on online platforms

In the Application distribution platforms users can buy, install software applications and give feedback on an app in the form of star ratings and reviews. Positive feedback promotes an app's popularity, the higher the app's rating, the higher it will be ranked on "Top lists", therefore it means that the app will reach a larger number of users, which in turn will increase the number of its downloads [Pagano & Maalej, 2013]. Positive user feedback impacts beneficially on customer acquisition, - positive ratings, number of downloads are the main factors that impact on rank of the app, customer conversion, - the content of the positive feedback may motivate users to pay for a premium version of an app, extra features, etc., and customer retention - positive feedback from other users can be perceived as a confirmation to keep using the same app.
Recent studies found that user reviews, so called a "voice of the users", can contain valuable information such as bug reports, feature and improvement requests, that can be useful for software developers to extract new software requirements, to direct the development efforts and to enhance upcoming releases [Pagano & Maalej, 2013], [Goul et al., 2012], [Jeong & Kim, 2021], [Guzman & Maalej, 2014].

However, Guzman & Maalej have identified some difficulties to analyze and use information from the user reviews:

1) vast range in the quality of the reviews - from innovative suggestions and useful advice to insulting reviews.
2) a "sentiment mix" in user reviews makes it hard to filter positive and negative feedback as well as the feedback for a specific feature.
3) star ratings don’t provide much information for the development team as rating shows the average evaluation of the whole app, but it doesn't show user evaluation of the specific feature. [Guzman & Maalej, 2014]

Pagano et al. analyzed user feedback which has information about applications in the AppStore and found that the most popular topic was praise (75% of all samples) [Pagano & Maalej, 2013]. They analyzed positive feedback that may contain useful information for the software requirements, and found that the feedback with the following patterns, such as {praise, feature request}, {praise, improvement request}, {praise, bug reports}, had a lower average rating

compared to other positive feedback patterns, and feedback with the above-mentioned patterns comprise approximately 7% of all feedback. In addition, their findings show that the following topics: feature requests, feature information, and praise were rated by users as the most helpful in feedback, but only 6% of all feedback were rated by users in terms of their helpfulness. We can see from the findings by Pagano et al. that the percentage of positive feedback that could be useful for software developers to improve software quality and to identify a new feature, is not significant, nevertheless, reviews that contain feature requests, feature information, and praise topics have been rated by users as the most helpful in determining the quality of the apps based on feedback from other users. Additionally, Pagano et al. detected that negative reviews are mostly "destructive" and lack user experience such as shortcomings and bug reports, which confirms the importance of positive reviews as a source for the app's evolution.
Lee & Santanam found that continuous quality updates as added desired features, fixed bugs that are based on the users' feedback as well as the high number of positive reviews have a positive impact on the freemium app's survival in the top charts [Lee & Santanam, 2014].
Also, many studies have analyzed user reviews in terms of sentiments to find useful information for requirement engineering. As previously mentioned, many studies identified a "sentiment mix" in user reviews, as one review may contain multiple topics, consequently, there can be as positive as negative sentiments in one user review. For this reason in our research we consider both positive and negative sentiments, as positive reviews may also contain negative sentiments as was demonstrated in research by Pagano et al., Jeong et al., and Goul et al. found that reviews with positive sentiments were much less informative than reviews with negative sentiments. Therefore, they suggest developers consider reviews only with negative sentiments, as they contain requests for new features written by real users and bug reports.
Additionally, De Haan and Menichelli determined that the sentiments and topics that customers have used in their feedback can help to predict future customer churn [De Haan & Menichelli, 2020].

Both approaches that were mentioned earlier to analyze user reviews in the application distribution platforms can help developers to learn customer pain points and satisfy their needs, by improving the application according to the crowdsourcing requirements.

User feedback can potentially affect a wide range of activities, as it promotes reachability of the software and by that acquires new customers, which in its turn can raise popularity of the software and raise the number of purchases and downloads of the software. User feedback also impacts a company's reputation, as the significant amount of positive feedback from real customers can make other customers trust this company. Sentiments used by the users in their feedback can also help to predict retention or churn of the customers. Also, feedback can be used as crowdsourcing to find new software requirements, upgrade software with the new version release. Moreover, user feedback can also be used as an indicator of software quality conformity.
The following figure demonstrates how positive feedback, as positive user review, rating, and word of mouth can affect the company's success.
Positive user reviews, positive word of mouth can motivate potential customers to purchase a product - customer acquisition. A product of good quality is more likely to satisfy customers, which can potentially build customer trust to a company of the purchased product. Positive experience from the purchased product will further raise the popularity of the product by the customer feedback, motivate further purchases of the product as well as it may raise an interest

in other products developed by the same company. Positive feedback as high ratings, positive reviews on other products of this company as well as the personal positive experience with the company may assure a customer to try other products by this company. Positive experience from the other products by the same company will build customer loyalty and therefore boost customer retention, that will generate recurrent revenues.

3.5.2 IS success model with user satisfaction dimension

Traditionally, a software development project is considered successful if it was delivered within schedule, budget, and with acceptable quality. However, in practice, software projects that meet the indicated factors but whose software system doesn't satisfy users, are more likely to fail, as well as the projects that exceed the budget or time constraints may still be successful if the end result corresponds or exceeds users' expectations [Bano et al., 2017]. That's why many research papers are devoted to identifying factors that influence success of the project and their interrelationship, such as user satisfaction and user involvement.

Many researchers defined user satisfaction as an indicator of the system's success. One of the most influential models, DeLone and McLean(D&M) IS success model, measures success with an amalgamation of six interrelated dimensions: system quality, information quality, use, user satisfaction, individual impact, and organizational impact.

"System quality" measures technical success, and it is focused on performance characteristics of the system: ease of use, system flexibility, system reliability, and ease of learning, as well as system features of intuitiveness, sophistication, flexibility, and response times [DeLone & McLean, 2016]. In the TAM Model, a close measure of the "system quality" is "perceived ease of use"(PEOU)[ibid: Urbach & Müller, 2011], [Davis & Davis, 1989].

"Information quality" measures semantic success, which is dependent on desirable characteristics from the system's output: relevance, understandability, accuracy, conciseness, completeness, understandability, currency, timeliness, and usability. This dimension is focused on obtaining good information quality that is produced by the system that should be useful for users. As a "system quality" dimension, "information quality" has a direct impact on user satisfaction.

"Service quality" measures the degree of the system support provided to the users of the system. The following characteristics facilitate the service quality: responsiveness, accuracy, reliability, technical competence of the IT staff.

"Use"/"intention to use" dimensions indicate the degree of the software system utilization by the users. Amount of use, frequency of use, nature of use, appropriateness of use, extent of use, and purpose of use can help to determine this dimension. The TAM model contributes more to the usage of the software system than the D&M model. By using two variables: "perceived ease of use" and "perceived usefulness", it determines the attitude towards use, the intention to use and the actual use of the system [ibid: Urbach & Müller, 2011]. In 2002, DeLone and McLean revised the D&M model, and updated it with the "intention to use" and other dimensions that will be discussed later. "Intention to use" was added to cover the use in other contexts. Moreover, they distinguish between the two, by defining "intention to use" as an attitude, and "use" as a behavior [Delone & McLean, 2002]. In the original D&M model "use" and "user satisfaction" are interrelated, and in the causal sense, greater "user satisfaction" will lead to increase in "use", with the new added dimension, high levels of "user satisfaction" will boost "intention to use" and as a result, "use" will be increased, too.

"User satisfaction" measures the extent to which a software system meets or surpasses user/customer expectations. Also, user satisfaction is considered as the main and mostly used measure of the IS success and adoption. As it was discussed earlier, "use", "intention to use" and "user satisfaction" are interrelated with each other, in a causal sense "user satisfaction" affects "intention to use", and that in its turn affects "use". High levels of "use" and "user satisfaction" increase "net benefits".

Proynova & Paech have also studied factors that influence on user feedback on predicted satisfaction with software systems, they have reported that prior familiarization with the good quality software system, leads to a positive expectation, and higher "predicted satisfaction", if the system performs better than expected, the "actual satisfaction" may be even higher (a positive disconfirmation), but in case if the system performs worse than expected, the "actual satisfaction" will be lower(negative disconfirmation). In the same way, prior familiarization with the bad quality software is likely to lower "predicted satisfaction", and the later disconfirmation has an impact on "actual satisfaction", which is known as the "second-grade effect," is one in which familiarity raises perceived understanding, which as the result raises "predicted satisfaction"[Proynova & Paech, 2013].

"Net benefits" measures the degree of the software systems contribution towards individuals' and organizations' success. The "individual" and "organizational impacts" dimensions of the original DeLone & McLean Model were combined together to form "net benefits". "Use" and "user satisfaction" are interrelated with the "net benefits". Some studies have suggested measuring "net benefits" with the financial metrics as return on investment, market share, productivity, profitability analysis, etc. Most of the studies that have used DeLone and McLean IS success model confirm the benefits that organizations and individuals derive from the use of software systems, and also reported a positive relationship between "software quality", "user satisfaction" and "net benefits"[ibid: Urbach & Müller, 2011].

In software development lifecycle user involvement was adopted to increase the user satisfaction and the user acceptance levels. This psychological mechanism of perceived control, by giving end users an opportunity to impact on the software implementation, makes future end-users more invested in the success of the software implementation and satisfied with the final product, which in turn will reduce their resistance to change and make them more willing to accept the new system [Bano et al., 2017], [Baronas & Louis, 1988].

Glass suggests a following metric to predict a user satisfaction from the product [Pressman, 2005]:

$$user\ satisfaction = compliant\ product + good\ quality + delivery\ within\ budget\ and\ schedule$$

3.6 Positive feedback mechanisms to integrate into the software development process

In order to respond on the second research question, which is: “How to integrate positive feedback into the whole process of software development?” we have decided to determine mechanisms that are used in SWE to promote positive feedback.

After positive feedback mechanisms integration, software team may perceive them as unfamiliar and artificial. However, regular usage of the mechanisms can facilitate the adoption of positive feedback in their work routine. As analysis of systematic literature review has shown, positive feedback has a positive impact on internal aspects of software engineers as mood, motivation, self-efficacy, job satisfaction, as well as on the external aspects as productivity, and performance. Consequently, integration of positive feedback mechanisms has not only positive impact on individuals, but it also positively impacts on the team, on the software product, reputation and popularity of the organization.

Systematic literature review has demonstrated the gap in positive feedback mechanisms. Mainly, all feedback mechanisms are targeted for collecting general feedback, but there are just few feedback mechanisms that are particularly targeted on positive feedback. They can be divided into 3 groups:

1) feedback mechanisms that used only to express positive feedback to endorse good work. The following mechanisms correspond to this group: job promotion, award bonuses and kudos cards.
2) feedback mechanisms that focused only on positive implications from the conducted events (OYA day (Out of Your Area) and Hackathon). If the results from these events are not positive, feedback won’t be negative, either. Moreover, those mechanisms can trigger fully positive feedback mechanisms, such as promotion or award bonuses.
3) feedback mechanisms where positive feedback is always provided, to highlight positive aspects, that should be repeated, but during retrospective and its varieties (as “lesson-learned”, starfish, 4Ls) negative feedback is also always noticed for improvements.

3.6.1 Fully positive feedback mechanisms

Kudos cards

Kudos cards is a technique, in which team members write praise to each other expressing the social recognition for a good work, but without financial reward [Henderson 2017]. There are no explicit requirements, as overtime work, extra work, etc., for providing this feedback, as well as the kudos cards can be granted unlimited number of times.

The usage and impact of the kudos cards:

In a work of [Rajeev K. Gupta et al. 2019], due to poor communication and collaboration between multiple cross-functional scrum teams, it was decided to integrate kudos cards mechanism to motivate engagement with other teams and to create a positive and appreciative work environment in an organization. As a result, this mechanism has motivated team experts to share knowledge, consult and collaborate with other scrum team members; moreover, the

kudos cards have boosted self-accomplishment feeling among the team members, filled everyone with good mood and energy, and contributed to the development of the team spirit in the organization.

Award bonuses and promotion

In some companies, positive feedback can be expressed by award bonuses nominated by a "peer" or a manager [Henderson, 2017]. Peer bonus is a financial reward for doing beyond the "call of duty", which can be awarded by a coworker only a limited number of times. Managers can also award their employees with bonuses, for e.g., as a spot bonus, which is rewarded "on the spot" for an achievement that deserves a special recognition, an annual performance bonus for reaching pre-established goals and benchmarks.

It is more common practice to express positive feedback on outstanding performance through promotions. As shows the study from [Nguyen et al., 2015] desire of being promoted motivates employees to put extra efforts to achieve performance goals, also promoted individuals feel great about themselves and thus, promotion opportunities can serve as both intrinsic and extrinsic reward

3.6.2 Feedback mechanisms that focused only on positive implications

Hackathon

Hackathon is a competitive event for which computer programmers and others (as interface designers, graphic designers, and project managers, etc.) form small ad-hoc teams and engage in short-term intense collaboration on software projects.

Hackathon aims to improve communication and collaboration channels within/outside a team, acquire new skills or expand existing ones, let to advance in a career. [A. Nolte et al, 2018]. And in some cases, as it was mentioned by a hackathon participant [A. Nolte et al, 2018] that "participation in the hackathon in [her/his] annual progress report regularly receives positive feedback by [her/his] manager", so the participation in the hackathon can be also considered as a positive impact on perception of management on hackathon participants' performance. Also, the same hackathon participant has stated that "participation in the hackathon changed her/his perception of the company in a positive way." [A. Nolte et al, 2018] After all above-mentioned arguments we can conclude that this mechanism is focused exclusively on positive implications from this event. Moreover, a hackathon can be a trigger of other feedback mechanisms as a promotion [A. Nolte et al, 2018] and award bonus. Therefore, a hackathon is considered as a positive feedback mechanism.

According to the research paper of [Rajeev K. Gupta et al. 2019], one of the pain points of the studied company was that cross-functional teams had become so self-sustained, that they were no longer interested in collaboration with other teams. In addition, there were several more problems that they wanted to solve with the help of the OYA day integration: 1) the scrum teams were working only on their own product, 2) the workload of the team members did not allow them to spend their time on implementing their own innovative solutions and ideas.

OYA day

OYA day is a one-day event, with the self-explanatory name, which is Out of Your Area. The main rules of this mechanism are to work on topics that are not from the team's project backlog and to work with people from other teams. Furthermore, teams should work on ideas that can potentially be finished during a day, such as reduction of compilation time or improvement in quality of logs, etc.

OYA day as a positive feedback mechanism

As the main purpose of the OYA day is to improve communication, develop a synergy between the teams, and try yourself in a new area, subsequently, we can assume that the final product achieved during this day wouldn't be strictly evaluated. As well as, in research by [Rajeev K. Gupta et al. 2019], only positive feedback and appreciation were mentioned as feedback on their achievements, it is tempting to think that feedback was only revealed, when it was positive. Therefore, OYA day can be counted as a positive feedback mechanism.

3.6.3 Feedback mechanisms that highlight positive and negative aspects

Project Management Institute (PMI) Project Management Body of Knowledge (PMBOK) defines lessons learned as knowledge gained from the process of conducting the project. This technique includes positive and negative aspects with the aim of promoting the positive aspects and preventing negatives. The classic question in "lessons learned" are: What was done well? What didn't go that well? What did you learn? Thus, this method can be used as a mechanism to gain positive and/or negative feedback from the team.

Retrospective and its varieties, as starfish and 4Ls have similar concepts to look back at past events, identify helpful changes to improve or amplify positive aspects to keep on doing or find pain points that should be stopped. Retrospectives help to point out potential benefits and improvements in ongoing projects, but it also helps to reduce stress, resolve conflicts, and motivate people.

The 4Ls is a retrospective technique where the team identifies what they loved, loathed, learned, and longed for in a project or sprint. This technique aims at fixing problems from the 'Loathed' list, keep doing aspects from the 'Loved' list, and 'Longed for' and 'Learned' lists help to develop ideas which actions should be taken in a future.

The starfish is a retrospective technique, that basically splits the canvas into 5 areas:

Keep doing – practices that the team is doing well and should continue doing.
Less of – something that has a low value or is not helpful for a project.
More of – highlight positive aspects, that should be done more.
Stop doing – something that is detrimental for a project and should be stopped.
Start doing – suggest new ideas, things to try.

Conclusion

This thesis is aimed to explore positive feedback in software engineering, as positive feedback is underestimated in the industrial practice and not much time and efforts are spent for providing positive feedback. Although our study indicates the importance to integrate and to practice positive feedback in the software development process in the regular basis.

During the systematic literature review, there were found theories from the psychological science field that amplify importance of feedback for individuals, which reflects on the atmosphere in team, produced software product, and the organization. The SLR provides better understanding of how positive feedback impacts on human aspects. We also have found factors that influence on the individual's perception of the feedback, which as the result help to predict whether positive feedback will have positive/negative impact or no impact at all. We also have found human aspects that get affected by the positive feedback, which are behavior, mood, job satisfaction, self-efficacy, motivation, performance.

Two research questions were determined based on the interviews from the prior research study. To answer the RQ1, we have derived metrics and best practices from the SLR that promote positive feedback within the software development lifecycle, by well-written requirements specification, efficient design, source code, testing. Moreover, during the SLR we have found human aspects that may also affect the objectiveness of the assessment of someone's performance, so the feedback provided by the source - "feedback sender" can be distorted.

For the RQ2, we have identified positive feedback mechanisms, feedback mechanisms that focus only on positive implications, and feedback mechanisms where positive feedback is always provided. As we believe that positive feedback mechanisms adoption leads to integration of positive feedback in the working routine, which as the result is beneficial for a software engineer, team, and organization. As we know from the systematic literature review that positive feedback not only enhances mood, motivation, self-efficacy, job satisfaction of a software engineer that has received positive feedback, but it also boosts higher productivity and performance of the team members.

Based on the findings of our SLR and the results from the interview analysis, we aim in future work to analyze factors in code review that impact on the performance assessment which negatively impact on feedback objectivity. Additionally, we aim to explore the Quantity of Recipients and the Quantity of Feedback Sources as feedback characteristics, as we believe that they impact on the individual's perception of the feedback.

Appendix

Original interviews

Interview 01

- Including different perspectives of different roles inside a project to get information about the work quality (quality of requirements)

[Teilnehmer_01 (74min 44s)]: Also was du vielleicht nochmal als Überlegung mitnehmen kann, ist dass all das was wir am Anfang tun mit Requirement und Qualitätsattributen am Ende des Tages ja dazu dienen entweder für sich selber rauszufinden, ob der Job gut erledigt ist oder auch dem Kunden oder dem Stakeholder beweisen zu können, dass die Arbeit gemacht und gut gemacht ist. So diesen Validierungs und Verifikations Aspekte vielleicht noch mal mit in die Überlegung mit einzubeziehen, weil das ist sozusagen noch mal ne zweite unterstützende Säule drunter. Einerseits hast du die ISO und das Schema mit den Qualitätsanforderungen, wo du sagst braucht man, dass das haben wir gelernte über die Jahrzehnte, dass wir Anforderungen in den Kategorien brauchen, aber du kannst ja auf der anderen Seite auch sagen, ich brauche Anforderungen in einer Art und Weise, wo ich danach auf Knopfdruck sozusagen sagen kann, ich habe meinen Job ordentlich gemacht. Der Aspekt der hilft dir vielleicht auch noch ein paar Schwächen oder Verbesserungspunkte in dem Requirements Prozess rauszufinden.

[Teilnehmer_01 (77min 14s)]: Wo wir auch häufiger auch in die Diskussionen mit den Requirements Engineering Prozess Leuten haben wir dann auch die Systemtester mit reingezogen und mal gesagt, wo sind denn im Systemtest die Schmerzen? Was zerreißt euren Kopf? Mit der Liste kann man dann auch dem Requirements Mensch Angst machen oder so oder halt entweder Angst oder ihm sagen "he pass auf da drüben dein Kollege, der heult fast und du könntest XYZ schon viel besser machen."

Interview 02

- Including different perspectives of different roles to know who needs what type of quality

[Interviewer (55min 25s)]: Dann würde ich auch mal auf diese systematisch Entwicklung zurückkommen, was hast du da in deinem Team so rein gebracht, wo du gemerkt hast? Ok? Da gibt's Probleme, wir müssen das jetzt immer systematischer angehen, was sind das so für Systeme, welche Vorgehen sind das?

[Teilnehmer_02 (55min 44s)]: Also ich habe da, schade dass ich es nicht zeigen kann. Ich habe das ganze in unserem Wiki ausgearbeitet dokumentiert, ich habe im Grunde mich eben mit einem Betroffenen abgestimmt ja, guck, was sind deren Schmerzen, habe dann auch denen ihre Anforderungen an einen Anforderungsmanagement Prozess aufgeschrieben. Also die ganzen Anwender eben miteinbezogen. Also meine Kunden sind die Entwickler, so gesehen. Um dann so verschiedene Perspektiven einzunehmen. Ein Entwickler ist nicht auch

gleich Entwickler, sondern Entwickler aus dem Teilprojekt Steuergerät, Hardware, Software, Funktion und so weiter sind auch alles unterschiedlicher Schlag Menschen habe ich auch festgestellt. Das ist tatsächlich so, ja. (Schmunzeln) Ich glaube die Kunst ist tatsächlich auf die ganzen Feinheiten einzugehen, weil du wirklich zu verstehen, wer ist dein Kunde und das ist ja auch ein wesentlicher Aspekt von Requirement Engineering. Also, ich wende eigentlich dass, was ich den Kollegen beibringen möchte, für mich in der Prozessentwicklung an. Habe Interviews geführt und zum einen gezielt also auch einer semi-strukturierten Form. Und habe natürlich auch in meiner tägliche Wechselwirkung mit den Kollegen und habe mir da ja auch drei Jahr ein Bild machen können. Und dadurch auch versucht eine kundenspezifische Lösung zu finden und neben den Entwicklern auch die Folge- und Schnittstellenprozesse zum Requirement Engineering sind auch super wichtig, das heißt, ich spreche eben auch mit einem Testmanager und mit dem Freigabe Manager und mit meinem Chef und mit meinem Chef Chef was sie sich von dem ganzen Thema für Antworten erwarten und bin im zu einem Bild gekommen. Also, wie schon gesagt, so, der Zweck von Requirement Engineering ist aus meiner Sicht, dass man einmal Kundenwünsche befriedigt, also Kundenwünsche erfüllt

- Example of positive feedback on requirement descriptions from a tester

[Teilnehmer_02 (69min 16s)]: [...] Ein Beispiel als wir letztens in so einer Coaching oder Systemlastenheftsession waren. Haben da einige Anforderungen umformuliert, es war relativ langwierig 6 oder 7 Anforderungen haben wir geschafft, als Systemanforderungen zu schreiben auf Basis von Kundenanforderungen und von denen wiederum Anforderungen an die Subsysteme abgeleitet und alles mit Traceability und Qualitätskriterien, also, ich sag mal 6 Systemanforderungen und dann noch mal jeweils fünf in den anderen Lastenheften also so ca. 15 - 16 Anforderungen. Hat zweieinhalb Stunden gedauert mit drei Personen, also ist ein ziemlicher Aufwand natürlich ist es erst mal neu und die Routine ist halt einfach noch nicht so da. Dann kam unser Testmanager als nächster rein und hat damit drauf geschaut und hat erst mal geschwiegen, weil wir dann noch fertig spezifiziert haben, wir wollten mit ihm noch über das Testing sprechen und er hat dann gesagt, ja, ich habe euch jetzt mal zehn Minuten schon zugehört und zugeschaut, wenn wir so Anforderungen schreiben, dann fällt mir Testmanagement in den Schoß. Das heißt, er muss da quasi nichts mehr machen, weil die Anforderungen testbar waren, wir haben Akzeptanz- und Verifikationskriterien gestellt, das wäre wahrscheinlich auch noch ein Attribut, was relevant ist gleich mit hinterlegt, wir haben eine Unterscheidung in funktionale und nicht funktionale Anforderungen gemacht, wir haben das atomar und ganze Latte an Qualitätskriterien spezifiziert. Ja, das heißt da merkst du, du kannst ja vielleicht auch diese Regel der Zehner-Regel oder Rule of ten, sag dir was?

Interview 06

- The project manager protects his team and gives them more time as positive feedback to continue the good work they have done so far.

[Interviewer (42min 47s)]: Feedback beispielsweise, wenn jetzt ein Requirements Ingenieur eine gute Anforderung schreibt, das kommt hinten beim Tester an und der Test sagt, okay cool, ich musste hier nicht nachfragen. Es war eine super Anforderung, weil meistens kommt da immer erst das Feedback zurück, wenn irgendwas schlecht war so, wenn man was positives zurückgekommen man dann einfach sagt okay, die Anforderung war gut, oder man

kriegt jetzt raus an der Anforderung wurde jetzt wenig rum gefragt, dass das auch mal wieder zurück gespielt wird und die Person sagt, cool, dass es auch motiviert.

[Teilnehmer_06 (43min 21s)]: Das glaube ich schon, dass das hilft. Also gerade bei dem, also wir bauen gerade so eine Produktplattform auf, da ist relativ viel Zeit, es gibt zwar von verschiedenen Seiten Druck, aber da legt der Projektleiter viel Wert drauf das korrekt gearbeitet wird und das wichtiger ist als unbedingt jede Zeitschiene, die ursprünglich mal geplant war, einzuhalten und steht da letztendlich auch vor dem Team und schirmt das dann bisschen ab und da hat man schon auch das Gefühl, wenn die Leute dass gut und hochwertig machen können und danach ist ja, ok hier das andere Projekt, dass es dann nutzt bereits diese Plattform und sagt ja, das funktioniert ja super. Dass das schon auch das Team motiviert. Das also ja, ich würde sagen das Hilft durchaus. Hilft reicht natürlich nicht, wenn am Ende einfach die Zeit ursprünglich gar nicht spendiert wurde, das ist ja auch eine Form von Feedback, wenn ich sage, ich lass dir die Zeit, weil ich weiß du machst das ordentlich und erspart und am Ende Zeit. Wie wir das angehen, das kommt ein bisschen drauf an, wie viel Einflussmöglichkeiten wir haben und wo das Problem steckt, [...]

- Positive feedback to convince persons who then pass it on to other people (for example to integrate changes)

[Teilnehmer_06]: [...] Ja, ich finde das mit dem Zurückspielen eigentlich auch noch eine ganz gute Idee und überhaupt mal dieses, das hat uns jetzt hier wirklich was gebracht, zu fördern. Also wir erzählen den Leuten schon, überlegt euch mal, wenn ihr jetzt eine Analyse macht für euren Change-Request, dann wollt ihr doch gerne auch sehen, wo habt ihr da Einfluss drauf und das wäre doch super hilfreich. Da kommt dann häufig ein, ja, richtig. Aber das ändert noch nicht unbedingt das Verhalten. Am wichtigsten ist es eigentlich oder sehr hilfreich ist es, meiner Erfahrung nach, wenn man ein paar Leute begeistern kann davon, die dann das auch, sozusagen, auch so Leuchttürme so ein bisschen tragen, ja, das ist super und ich erzähle das auch wieder weiter meinen Leuten und versuche selber. Also falls man da so ein paar Leute gewinnt an günstigen Stellen, die davon überzeugt sind, dann wird das auch besser. Wie gesagt, großer Punkt, ist immer die Projektleitung und Zeitdruck von der Seite. Wenn da einfach mit unrealistischen Zeiten gerechnet wird und die Leute am Ende nur danach beurteilt werden, ob da eine Software geliefert wurde oder nicht und weniger danach, das sind wir aber wieder beim Feedback, ob hinten die Dokumentation auch ordentlich war, dann werden sie sich natürlich daran halten und eine Software liefern. Und dann fällt halt das runter auf das man da evtl. nicht achtet.

- Convincing persons and provide tooling for positive feedback

[Teilnehmer_06 (74min 13s)]: Ich glaube der [menschliche Aspekt] ist überall im Softwareengineering drin. Der ist unheimlich präsent und unheimlich wichtig, also deswegen Leute überzeugen, mitnehmen, solche Techniken, auch das mit dem Feedback, ob man dass dann mit einem Tool unterstützt werden kann, dass die auch mitkriegen, dass das unten

jemand benutzt. Das ist, glaube ich, auch in jeder Form sehr sehr wichtig [...]

Interview 08

- Providing feedback by doing a “lesson learned”

[Teilnehmer_08 (47min 29s)]: Also, ich glaube so eine Art Lessons learned im nachhinein. Wäre schon hilfreich. Wo mir auch gerade einfällt gerade so lessons-learned werden ja eigentlich gefühlt zu jedem Thema gemacht, also so jeder Entwickler Runde macht dann ein lessons learned und jeder Architekturrunde oder so oder bei Requirements kenne ich das so noch nicht. Das werde aber wahrscheinlich schon im Nachhinein betrachtet keine schlechte Idee, ich habe natürlich jetzt selber auch noch nie in einer RE Abteilung an sich gearbeitet und weiß nicht wie der Alltag jetzt genau aussieht, aber ob sie sowas tun weiß ich nicht, sonst würde man wahrscheinlich in der Literatur was dazu finden.

- Improving job satisfaction. For example, the feedback can be provided by a scale or a flag for each requirement inside the requirements management tool.

[Teilnehmer_08 (49min 56s)]: Finde ich eine sehr interessante Idee [positives Feedback] vor allem weil (...) Also, das ist ja glaube, also das ist ja ja wahrscheinlich auch ein bisschen allgemeines Thema positives Feedback, glaube ich, eh so nicht nur jetzt auf Anforderungen bezogen, so ein Thema in vor allem in großen Firmen, da kommt das ja sehr häufig vor, dass ich Leute zu wenig gewertschätzt fühlen. Speziell darauf bezogen ist es interessant. Weil ich mir vorstellen könnte, dass dann ja vielleicht der Job ein bisschen interessanter vielleicht wird, also ein bisschen lieber gemacht wird, sage ich jetzt mal, ich habe das Gefühl so insgesamt, dass Anforderungen schreiben oder Anforderungsmanagement ein bisschen an schlechten einen schlechten Ruf hat, so, ja man muss das ja nur abtippen. So dann ist es immer nicht richtig, so gefühlt jetzt und das wäre natürlich dann schon eine Möglichkeit diesen Job auch ein bisschen vielleicht in einem anderen Glanz, sage ich jezt mal, erscheinen zu lassen. Ich muss gerade drüber nachdenken, wie dieser Weg aussehen könnte, weil es ist natürlich auch immer der Entwickler kriegt halt relativ einfach gutes oder schlechtes Feedback, weil entweder das was er entwickelt hat, stimmt oder er kriegt einen Bug Fix also einen Bug zurück, so nach dem Motto, sage ich jetzt mal, ich versuche mir gerade vorzustellen, wie das für Requirements Engineere ausschauen könnte oder für eine Anforderung ausschauen könnte, wahrscheinlich müsste man sowas oder ich stelle mir gerade sowas vor wie, das vielleicht so eine Anforderung die dann der Entwickler oder der Architekt bekommt und dann der Entwickler bekommt, dass ich vielleicht sowas hat wie ein Approved oder so als Flag vielleicht, also diese ganzen Tools haben ja hunderte Felder dann wenn so irgendwie so ein Ding mal beim Entwickler landet und der hat das fertig und sag dann approved mit dem Ticket konnte ich gut arbeiten z.b. und mit dem Ticket bin ich nicht zurecht gekommen oder so. So versuche ich mir das gerade vorzustellen, weil das muss natürlich auch irgendwie immer in einen Alltag reinpassen. Also in einem Arbeitsalltag sage ich mal, weil ich kann ja nicht nach jedem Dingen dann aufstehen und zu dem Requirements Engineer laufen und dann sagen hey, das Ticket hat gepasst oder das Ticket hat nicht gepasst, oder so. Könnte ich also vorstellen könnte ich mir das schon, ich weiß nicht, ob das ob das praktikabel

ist, aber es ist ein interessanter Ansatz.

[Interviewer (52min 39s)]: Genau, das ist nämlich eigentlich auch der Punkt. Aber das ist so ein bisschen das wo die Interviews halt sehr stark darauf hinauslaufen, dass halt das Bewusstsein nicht da ist und die Frage ist, wie kann man das Schüren [...]

[Teilnehmer_08 (54min 22s)]: Ja, verstehe ich. Also ich habe selber jetzt auch dazu noch nichts gehört, also man kennt dieses klassische ok, man hat jetzt Satzschablone nicht eingehalten oder so und kriegt dann schlechtes Feedback, aber so, wie sie beschrieben haben, das habe ich noch nicht gehört und finde ich wirklich interessant. Ich versuch gerade wirklich mir vorzustellen, wie man das auch in einen Arbeitsalltag also wie man jetzt sagen, also, man könnte sowas ja auch, wie soll ich sagen, selbst gesteuert dann in ein Tool z.b. implementieren, also jetzt gerade so ein Tool wie Polarion ist unfassbar konfigurierbar und da jetzt zu sagen ein Flag oder eine Bewertung rein zu also zum Dropdown von eins bis zehn meinetwegen oder so rein zu machen, wäre hat überhaupt kein Problem für das Tool an sich die Frage ist, natürlich dann immer nach der Arbeitsweise dann auch, das muss halt immer in die Leute und ich glaube, das ist auch so ein Thema was sich so schwer bewerkstelligen lässt, das ist ja nicht nur die Methode dafür geben muss, sondern dass es halt auch bei den Leuten immer so im Kopf dann ankommen muss. Also die Arbeitsweise sage ich mal.

Interview 10

- Show the result to stakeholders who have requested this particular feature (strengthen customer loyalty)

[Teilnehmer_10 (56min 53s)]: Viel zu selten. Also ich tatsächlich bin auch ein Fan Dinge die ohne Widerstände und gut gelaufen sind sichtbar zu machen, aber tatsächlich sind wir in unserer Gesellschaft und auch der agile Ansatz, der ist so problemorientiert, dass heißt, du bist in einem stetigen Verbesserungszyklus, also das heißt du guckst ja eigentlich immer Probleme an und immer das was nicht gut geklappt hat und du guckst sehr selten an, was deine Stärken sind, oder was sehr gut funktioniert hat und Retrospektiven wird es von hinten von Scrum Mastern zwar immer wieder eingebaut, dass sich Mitarbeiter quasi auch da stärken können, aber auf Prozessualer Hinsicht oder so, wird sowas eigentlich kaum sichtbar gemacht. Da geht's eigentlich immer um Prozessverbesserungen und das Sichtbarmachen von Problemen im Prozess.

[Interviewer (57min 49s)]: Wie würden Sie jetzt vorgehen, wenn sie sich da was überlegen müssen positive Ergebnis eher vorzustellen, gerade auch in Bezug auf Anforderungen?

[Teilnehmer_10 (58min 6s)]: Ja, also so ein Muster Positivbeispiel, was ich mir vorstelle ist bezugnehmend vielleicht auf den Anfang unseres Gesprächs, ich habe einen eine bestimmte Lösung, ob es eine Software ist oder eine Spezialität auf den Markt gebracht und ich weiß sogar noch welche zehn Kunden mir gesagt haben, das muss unbedingt gemacht werden und wenn ich jetzt die Möglichkeit habt, diese zehn Kunden noch einmal gezielt drauf zu stoßen zu sagen, ey Leute das hier ist gerade heute live gegangen. Guckt euch das mal an, wie findet ihr das? Und die sagen mir, mensch, das ist ja super, das ist genau das was ich wollte, dann ist es doch uns gelungen von der ersten Äußerung eines Bedürfnisses bis hin zur Auslieferung

mindestens zehn Menschen und vielleicht noch viele weitere glücklich zu machen und das ist für mich etwas das man unbedingt sichtbar machen sollte und dadurch natürlich auch. Ja, darstellen kann wie wertvoll es ist, dann etwas so durchgängig auch tracken zu können, weil man damit eben einfach einfach einen direkten Kundenbezug viel besser herstellen kann und damit auch eine Verbundenheit des Kunden mit dem Unternehmen und mit der Software pflegen kann und ich glaube dieses Thema Verbundenheit von Kunden mit Software ist zukünftig schon ein bildlicher Schlüsselfaktor, um ja Kunden begeistern zu können und auch an Software emotional auch an Software binden zu können. Das ist, denke ich, schon vergleichbar mit heute mit dem Android und Apple Thematiken, es gibt zwei Fraktionen und wahrscheinlich gebe ich sie sich beide nicht viel, aber es ist doch eher selten, dass der eine bewusst in das System des anderen wechselt, weil doch eine sehr starke Verbundenheit mit dem jeweiligen System existiert.

- Highlighting positive aspects

[Teilnehmer_10 (60min 53s)]: Naja, letztendlich. In den ganz normal Entwicklungszyklus eines Teams, also wenn ein Team beispielsweise eine Anforderung bekommt und sich entscheidet wie wird das gemacht, dann sollte man vielleicht da zumindest schon bevor der erste Federstrich gemacht wird noch mal reflektieren, haben wir eigentlich alle verstanden, was wir jetzt machen sollen und da könnte man eben dann mal nicht nur problemorientiert ran gehen und sagen 80% habe ich verstanden, aber die 20% nicht und da müssen wir jetzt noch mal ran, sondern man kann natürlich auch mal sagen, aber ja super 80% habe ich schon mal verstanden, das ist besser als bei vielen anderen Anforderungen die ich in der Vergangenheit bekommen habe, also klar sowas kann man natürlich machen, man muss aber halt das explizit einfordern, weil sonst im Alltag des einfach untergeht. Wie gesagt, wir sind da sehr problemorientiert arbeiten wir da und deswegen wird eigentlich nur Bezug genommen auf das was nicht passt.

Interview 11

- Having a “thumbs up” button to provide fast feedback on a requirement. It is important that it doesn’t take too much time to provide feedback. Alternatively, explicit meetings for feedback can be arranged.

Also, ich finde sowas wie ein Feedback Button, sage ich jetzt mal, eigentlich immer ganz cool oder einfach nur so ein so ein Rating irgendwie es muss ja wirklich nicht viel sein, wie du auch sagst. Einfach nur ein, hat dir diese Anforderung geholfen, ja, nein, wäre ja schon meistens hilfreich, ne, oder? Weiß nicht wie sich sowas in einem Tool oder in einem Prozess gut abbilden lässt, das ist halt die Frage. Wie du sagst, es darf die tägliche Arbeit nicht noch belästigen oder behindern, da sind dann so Termine einfach meistens noch mal sinnvoller, dass man sich dann explizit für Zeit nimmt oder sagt, man holt sich aktiv Feedback ein sowas, so wird ich halt persönlich machen, ich glaube, also gerade uns in der Firma ist halt schwierig, solche Bewertungssysteme zu etablieren, weil dann hast du immer noch den Betriebsrat mit auf der Matte. Der sagt, was, das ist eine Mitarbeiterbewertung. Das geht gar nicht, aber vom Prinzip her, glaube ich, würde ich es wirklich lieber selber aktiv einfordern im Sinne von Hey, gib mir doch mal bitte dazu Feedback, wie kann ich besser machen? Aber sonst finde ich so einen Daumen runter, Daumen hoch, schon ganz gut. [Interviewer (27min 1s)]: oder vielleicht

doch eher nur Daumen hoch, dass man sagen kann. Okay, dann dann dann will man jetzt ja niemanden hier schlecht machen so ungefähr ja, okay genau, weil dass das war nur noch mal so ein Punkt der aufploppte und das kam auch in vielen Gesprächen auf und es war nicht halt eigentlich super interessant noch mit reinzubringen, auch wenn es nicht so direkt mit Vollständigkeit, Konsistenz und Traceability zu tun hat. [Teilnehmer_11 (27min 30s)]: Ja, das stimmt, aber das hilft ja auch den REs sich selber zu verbessern. Also besser zu werden bei der Formulierung der Anforderungen.

- Regularly integrate positive feedback within the meetings. Regular feedback may depends on the team size and the culture of communication.

Ich glaube tatsächlich das fehlt einfach, so ein bisschen auch diese regelmäßige Feedback zwischen den, im Sinne von, he schau mal super, aber das könntest du noch besser machen, da fehlt mir jetzt aber auch noch ein bisschend er Input, aber auch hier, das ist super, da könntest du dich auch zukünftig orientieren. Also das sehe ich genauso, da fehelt es glaube ich auch einfach noch. Ich glaube in manchen Teams auch wieder hier, je nach dem wie groß es auch ist, da ist die Kommunikation schon besser, aber ich glaube gerade in größeren Teams bin ich da der gleichen Meinung, da fehlt das.[Interviewer (24min 40s)]: Das entstand auch so ein bisschen daraus aus der Aussage: nicht kritisiert ist genug gelobt. [Teilnehmer_11]: Das ist in den Franken verbreitet ja.

Interview 12

- Positive feedback is very rare. It may improve the feeling and the motivation of individuals, but it may not change anything related to long term goals

[Teilnehmer_12 (36min 49s)]: Also die Erfahrung zeigt, mit Lob wird gespart. [Lachen] im Regelfall würde, wenn das Ergebnis passt, dass schon zurückgespielt bekommen. Aber im eigentlichen Prozess hast du ganz selten ein Lob oder das, ja war gut, war super. Das häufigste ist dann doch da passt es nicht und da passt es nicht, da jeder seine Sache fertig stellen will und die Hemmnisse überwiegen dann quasi.Spricht man legt dann doch lieber, die Finger in die Wunde, du hast da was vergessen, dort brauche ich noch die und die Info, anstatt zu sagen, es war super.[Interviewer (37min 28s)]: Und würde sowas aber auch die Arbeitsresultate oder deine Arbeit verbessern oder ist es mehr sage ich meine psychologische Aspekt, dass man auch mal soll ich was Positives erfährt? [Teilnehmer_12 (37min 46s)]: Ich denke, es ist mehr der psychologische Aspekt, dass man sich besser fühlt, mehr wertgeschätzt fühlt und seiner Tätigkeit. Ja und wahrscheinlich motiviert sich vielleicht auch ein bisschen mehr, vielleicht mehr nach links nach rechts zu schauen, aber ich denke das sind oftmals auch kurzfristige Momente, aber fürs langfristige, glaube ich, würde es das kaum ändern. Mehr das eigene Wohlbefinden als dass die die Arbeit merklich darunter wirklich verbessert werden könnte. Wichtig ist eher der wertschätzende Umgang mit den Kollegen, die den Finger in die Wunde legen, ich glaub das ist das entscheidende, ob du beim nächsten mal noch mit ihm reden möchtest oder der sagt dann komme, so wie du mich angegangen bist, da hör mir auf.

Ich glaube das ist das entscheidende an der Sache.

Interview 13

- Description of a scenario where the RE adjusted the template of the requirements and gets positive feedback.

[Teilnehmer_13 (56min 44s)]: Ja, und zwar wir haben ja immer eine Retro alle zwei Wochen. Wir haben ja vor zwei Wochen habe das mit den Tickets wieder ein bisschen umgestellt und ich habe schon in den Gespräch mit den Softwareentwicklern unter der Woche gemerkt den gefällt das, die haben auch gesagt, he cool, da können wir ja sogar nach 2 Wochen schon was ausliefern. Die Stories sind so beschrieben, dass ich den Suchschlitz schon ausliefern könnte. Der Kunde kann damit zwar noch nichts machen, aber der ist schon drin, der ist implementiert. Und dann in der Retro haben uns die verschwiegenen Softwareentwickler, da hat er eine, ich sage mir immer, wenn ich den einen überzeugt habe, dann gehen die anderen sowieso alle mit, weil der ist nämlich so ein harter Brocken und der hat dann gesagt, das war richtig gut, der hat richtig ausdrücklich von sich selbst auch gesagt, das war super. Genauso brauche ich das und das hilft mir total, weil ich weiß, ich bin auf dem Weg, aber ich hatte schon ein gutes Gefühl, weil in den ersten Terminen, wo ich die Umstellung präsentiert habe, habe ich schon gemerkt das gefällt ihnen, so wie es ist, weil die hatten da auch nichts mehr anzubessern. Haben immer wieder gelobt. Ja, genau, so können wir super arbeiten. Aber das ist dann auch eine Feedbackkultur im Team ne, also ich will jetzt mal sagen, bei uns ist das alles sehr offen, sehr direkt, sehr freundlich, sehr wertschätzend im Team. Also, wir arbeiten einfach alle so, ich Lob auch wo es geht, na und von daher sind keine stillschweigenden Menschen bei uns, das passt.

- Reusing ideas or sketches (of for example the frontend developer) inside the requirement ticket recognizes the work and is, therefore, a form of positive feedback

Zum einen diesen Screendesign erstellt ja bei uns der Frontend-Entwickler z.b. und wenn der seine Sachen wiederfindet in meinen Tickets, dann freut der sich natürlich und bringt das direkt in Zusammenhang, ne? Das ist schon mal ein Vorteil. Zum anderen, ja, die haben einfach gesagt, ist geil. Also als ich alles Umgestellt habe, dachte ich, oh Gott, jetzt habe ich wieder alles umgestellt. Ich glaube die töten mich. Und dann sagen die, das ist geil, das ist genau dass, was wir wollten. Man muss so ein bisschen in Vorleistung immer gehen. Bei denen, weil Softwareentwickler, habe ich feststellte, das sind so Menschen, die sagen nicht sofort und direkt was Sie brauchen und was sie wollen, aber man merkt ihnen an der Nase an, dass irgendwas nicht passt an den Tickets, aber sie sagen es nicht und ich bin da sehr feinfühlig bei sowas und denke irgendwas passt denn jetzt nicht und ich fragte dann, ist das richtig, ist das richtig, ist das richtig und da habe ich so ein bisschen für mich selber herausgefunden, ich muss die einfach Fragen direkt, passt dir dieser Satz. Gehst du da mit, ist es auch das, haben wir das selber Verständnis dafür und dann merkt man schon, dann gehen die langsam auf und sagen, naja, nee ich würde das anders schreiben und so nähert man sich an und bekommt als RE ein Gefühl dafür, wie wollen die denn die Stories haben, weil die sagen gar nix und das klappt eigentlich bei uns ganz gut, wenn ich dann die Bilder mit einbringe und dann also ich beschreibe die Tickets ja immer aus fachlicher Sicht, also aus Kundensicht so ein bisschen mit ganz einfachen Wörtern, also ich verwende da wender die

fetten Fachbegriffe noch irgendwelche technischen Begriffe, die kenne ich gar nicht. Die haben festgestellt, dass das gut für sie ist, wenn das so einfach beschrieben ist, weil aus technischer Sicht dann Ihnen viele Möglichkeiten noch offen bleiben, wie sie es umsetzen, weil würde ich technisch, wovon ich überhaupt keine Ahnung habe, schon vorgegeben, wie sie es zu tun haben, würde ich schon wieder in ihren Bereich so ein bisschen mit reingehen, was sie nicht wollen wahrscheinlich und das ist das hat bei uns eine Weile gedauert, also ich glaub so ein-einhalb bis ein Jahr bis ich wusste, wie wollen sie es denn die Herren, also das habe ich nur durch, also durch meine Person an sich, dass ich da halt bohre. Ich merke es passt nicht und dann bohre ich nach, bei jedem einzelnen, ist das so richtig, irgendwann sagen sie es dann schon, wie sie es anders haben. Die wollen hat auch niemanden kränken, ne, also viel reden.

References

Abdel-Hamid, T. K. (1989). A Study of Staff Turnover, Acquisition, and Assimilation and Their Impact on Software Development Cost and Schedule. Journal of Management Information Systems, 6(1), 21–40. http://www.jstor.org/stable/40397903

Amutan, K.I. (2014) “A Review of B.F. Skinner’s ‘Reinforcement Theory of Motivation,’” Int. J. Res. Educ. Methodol..

Bacchelli A. and Bird, C. (2013) “Expectations, Outcomes, and Challenges Of Modern Code Review,” in ICSE, 2013, pp. 712–721.

Baldwin, C.Y.; Clark, K.B. (2000). "Chapter 3: What Is Modularity?". Design Rules: The power of modularity. MIT Press. pp. 63–92. ISBN 9780262024662. Retrieved 18 May 2018.

Bandura A. (1977). Self-efficacy: toward a unifying theory of behavioral change. Psychological Review.84(2):191-215. https://doi.org/10.1037/0033-295X.84.2.191 PMID: 847061.

Bandura A. (1986). Social foundations of thought and action: A social cognitive theory. Englewood Cliffs, NJ: Prentice-Hall.

Bandura A. (1997). Self-efficacy : the exercise of control. W.H. Freeman and Company.

Bandura, A. (1991). Social cognitive theory of self-regulation. Organizational Behavior and Human Decision Processes, 50(2), 248–287. https://doi.org/10.1016/0749-5978(91)90022-L.

Bano, Muneera & Zowghi, Didar & da Rimini, Francesca. (2017). User Satisfaction and System Success: An Empirical Exploration of User Involvement in Software Development. Empirical Software Engineering. 22. 10.1007/s10664-016-9465-1.

Baronas, A.-M. K., & Louis, M. R. (1988). Restoring a Sense of Control during Implementation: How User Involvement Leads to System Acceptance. MIS Quarterly, 12(1), 111–124. https://doi.org/10.2307/248811

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., & Jeffries, R. (2001). Manifesto for Agile Software Development.

Beecham S., Baddoo N., Hall T., Robinson H., and Sharp H. (2008). Motivation in Software Engineering: A systematic literature review. Inf. Softw. Technol. 50, 9–10 (August, 2008), 860–878. https://doi.org/10.1016/j.infsof.2007.09.004

Boehm, B.W. (1984) "Verifying and Validating Software Requirements and Design Specifications," in IEEE Software, vol. 1, no. 1, pp. 75-88, Jan. 1984, doi: 10.1109/MS.1984.233702.

Boehm, B.W. (2001). Software Engineering Economics. In: Broy, M., Denert, E. (eds) Pioneers and Their Contributions to Software Engineering. Springer, Berlin, Heidelberg.

https://doi.org/10.1007/978-3-642-48354-7_5

Bosu, A., Carver, J. C., (2014) “Impact of Developer Reputation on Code Review Outcomes in OSS Projects: An Empirical Investigation,” in ESEM, 2014, pp. 33–42.

Bosu, A., Carver, J. C., Bird, C., Orbeck, J., and Chockley, C. (2017) “Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft,” TSE, vol. 43, no. 1, pp. 56–75.

Buffardi, Kevin and Edwards Stephen H. (2014). Responses to adaptive feedback for software testing. In Proceedings of the 2014 conference on Innovation & technology in computer science education (ITiCSE '14). Association for Computing Machinery, New York, NY, USA, 165–170. https://doi.org/10.1145/2591708.2591756

Carayon, P., Hoonakker, P.L., Marchand, S., & Schwarz, J. (2003). Job characteristics and quality of working life in the IT workforce: the role of gender. SIGMIS CPR '03

Chen, Ling-Hsiu. (2008). Job satisfaction among information system (IS) personnel. Computers in Human Behavior. 24. 105-118. 10.1016/j.chb.2007.01.012.

Chillarege, R. (1999). Software Testing Best Practices.

Cito, J., Wettinger, J., Lwakatare, L. E., Borg, M., and Li, F. (2018) "Feedback from operations to software development—A DevOps perspective on runtime metrics and logs", Proc. Int. Workshop Softw. Eng. Aspects Continuous Develop. New Paradigms Softw. Prod. Deployment, pp. 184-195, 2018.

Dabbish, L., Stuart, C., Tsay, J. and Herbsleb, J. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (New York, NY, USA, 2012), 1277–1286.

Davis, F. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Quarterly. 13(3), 318–340.

Davis, F. D., Bagozzi, R. P., & Warshaw, P. R. (1989). User acceptance of computer technology: A comparison of two theoretical models. Management Science, 35(8), 982–1003.

Dawood, Yasir & Hashim, Nor Laily & Md Rejab, Mawarny & Romli, Rohaida & Haslina, Haslina. (2017). Coverage criteria for test case generation using UML state chart diagram. AIP Conference Proceedings. 1891. 020125. 10.1063/1.5005458.

De Haan, E., & Menichelli, E. (2020). The incremental value of unstructured data in predicting customer churn. MSI Working Paper Series, 20(105), 1-49.

DeLone, W., and McLean, E., (2016), "Information Systems Success Measurement", Foundations and Trends® in Information Systems: Vol. 2: No. 1, pp 1-116. http://dx.doi.org/10.1561/2900000005

Delone, William & McLean, Ephraim. (2002). Information Systems Success Revisited (PDF).

Hawaii International Conference on System Sciences. 8. 238. 10.1109/HICSS.2002.994345.

Demirors, E., Sarmagik, G., Demirors, O. (1997) The role of teamwork in software development: microsoft case study, in: Proceedings of the EUROMICRO Conference, pp. 129–133.

Dictionary.com, 2022. Feedback. Retrieved 2022, from Dictionary.com website: https://www.dictionary.com/browse/feedback

Eden, Dov. (1992) "Leadership and expectations: Pygmalion effects and other self-fulfilling prophecies in organizations", The Leadership Quarterly, Volume 3, Issue 4, Pages 271-305, ISSN 1048-9843, https://doi.org/10.1016/1048-9843(92)90018-B.

Faragher, E.B., Cass, M. and Cooper, C.L., (2013). The relationship between job satisfaction and health: a meta-analysis. From stress to wellbeing Volume 1, pp.254-271.

Fenton N. and Bieman J. (2014). Software Metrics: A Rigorous and Practical Approach, Third Edition (3rd. ed.). CRC Press, Inc., USA.

Fitzpatrick, R. (2000). Software Quality : Definitions and Strategic Issues Copyright © 1996.

Fitzsimmons, Ann and Love, Tom. (1978) A review and evaluation of software science. Computing Surveys, 10(1), March 1978. https://doi.org/10.1145/356715.356717.

Flater, D.W. (2018). 'Software Science' revisited: rationalizing Halstead's system using dimensionless units.

França, A. C. C., Gouveia, T. B., Santos, P. C. F., Santana, C. A., and da Silva, F. Q. B. (2011) "Motivation in software engineering: A systematic review update," 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), 2011, pp. 154-163, doi: 10.1049/ic.2011.0019.

Goul, M., Marjanovic, O., Baxley, S., Vizecky, K. (2012). Managing the enterprise business intelligence app store: Sentiment analysis supported requirements engineering. In: Proceedings of the 45th Hawaii International Conference on System Sciences (HICSS), pp. 4168–4177

Graziotin, D., Fagerholm, F., Wang, X., Abrahamsson, P. (2018) “What happens when software developers are (un)happy”, Journal of Systems and Software, Volume 140, 2018, Pages 32-47, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2018.02.041.

Graziotin, D., Wang, X., and Abrahamsson, P. (2014) “Software developers, moods, emotions, and performance,” IEEE Software, vol. 31, no. 4, pp. 24–27, 2014.

Gupta, R.K., Jain, S., Singh, B., Jha, S.K. (2019) Key factors in scaling up agile team in matrix organization. In: Proceedings of the 12th Innovations on Software Engineering Conference (formerly known as India Software Engineering Conference); 2019. p. 1–5.

Guzman E. and Maalej W. (2014) "How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews," 2014 IEEE 22nd International Requirements

Engineering Conference (RE), 2014, pp. 153-162, doi: 10.1109/RE.2014.6912257.

Hackman, J. R., & Oldham, G. R. (1976). Motivation through the design of work: Test of a theory. Organizational Behavior & Human Performance, 16(2), 250–279. https://doi.org/10.1016/0030-5073(76)90016-7

Halstead, M. (1977). Elements of Software Science (Operating and programming systems series). Elsevier Science Inc., USA.

Heitlager, Ilja & Kuipers, Tobias & Visser, Joost. (2007). A Practical Model for Measuring Maintainability. 30 - 39. 10.1109/QUATIC.2007.8.

Henderson, F. (2017). Software engineering at Google. Technical report, Google, 2017. URL: https://arxiv.org/pdf/1702.01715.pdf

Herzberg, F., Mausner, B., and Snyderman, B. (1959) 'The Motivation to Work.' 2nd ed, John Wiley & Sons Inc.

IEEE Recommended Practice for Software Requirements Specifications. IEEE Standard 830-1998 (R2009), Institute of Electrical and Electronics Engineers. (2009).

Inamori, T. and Analoui, F. (2010), "Beyond Pygmalion effect: the role of managerial perception", Journal of Management Development, Vol. 29 No. 4, pp. 306-321. https://doi.org/10.1108/02621711011039132

Jansen, H. (2010). The logic of qualitative survey research and its position in the field of social research methods. In Forum Qualitative Sozialforschung/Forum: Qualitative Social Research (Vol. 11, No. 2).

Jeong, J., Kim, N. (2021) Does sentiment help requirement engineering: exploring sentiments in user comments to discover informative comments. Autom Softw Eng 28, 18. https://doi.org/10.1007/s10515-021-00295-w

John E. Bentley, Wachovia Bank, Charlotte NC, ―Software Testing Fundamentals—Concepts, Roles, and Terminology, SUGI 30.

Jongeling, R., Datta, S., and Serebrenik, A. (2015)"Choosing your weapons: On sentiment analysis tools for software engineering research," in IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 531–535.

Kalliamvakou, Eirini & Bird, Christian & Zimmermann, Thomas & Begel, Andrew & Deline, Robert & German, Daniel. (2017). What Makes a Great Manager of Software Engineers?. IEEE Transactions on Software Engineering. PP. 1-1. 10.1109/TSE.2017.2768368.

Kaufmann, G. (2003) Expanding the Mood-Creativity Equation, Creativity Research Journal, 15:2-3, 131-135, DOI: 10.1080/10400419.2003.9651405

Kaur, R., Chacal, K.K. (2021) "Exploring factors affecting developer abandonment of open

source software projects".

Kenyon, George & Sen, Kabir. (2015). The Perception Process. 10.1007/978-1-4471-6627-6_5.

Khan, I.A., W.-P. Brinkman, W.-P., Hierons, R. M. Do moods affect programmers' debug performance?, Cognition, Technology & Work 13 (2011) 245–258.

Kim, S., and Wright, B. (2007) 'IT Employee Work Exhaustion.' Review of Public Personnel Administration 27, no. 2, pp. 147-170.

Kitchenham, B. (2004). Procedures for performing systematic reviews, Keele, UK, Keele University 33 (2004) 2004.

Kluger, A., DeNisi, A. (1996) The effects of feedback interventions on performance: a historical review, a meta-analysis, and a preliminary feedback intervention theory. Psychol. Bull. 119(2), 254–284 (kuhn2017, P. 7: 2874)

Kohli, A.K., Jaworski, B.J. (1994), "The Influence of Co-Worker Feedback on Salespeople," Journal of Marketing, 58(October), 82–94.

Kulhavy, R.W., Wagner, W. (1993) Feedback in Programmed Instruction: Historical Context and Implications for Practice;. In J. V. Dempsey & G. C. Sales (Eds.), Interactive instruction and feedback (pp. 3–20). Englewood Cliffs, NJ: Educational Technology.

Lee, Gunwoong & Santanam, Raghu. (2014). Determinants of Mobile Apps' Success: Evidence from the App Store Market. Journal of Management Information Systems. 31. 133-170. 10.2753/MIS0742-1222310206.

Lee, M.S.M., Lee, M.B., Liao, S.C. and Chiang, F.T., (2009). Relationship between mental health and job satisfaction among employees in a medical center department of laboratory medicine. Journal of the Formosan Medical Association, 108(2), pp.146-154.

Lee, Ming-Chang. (2014). Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance. British Journal of Applied Science & Technology. 4. 10.9734/BJAST/2014/10548.

Locke, Edwin & Latham, Gary. (2002). Building a Practically Useful Theory of Goal Setting and Task Motivation: A 35Year Odyssey. American Psychologist - AMER PSYCHOL. 57. 705-717. 10.1037/0003-066X.57.9.705.

Locke, Edwin A. (1968), "Toward a theory of task motivation and incentives." Organizational Behavior and Human Performance 3: 157-189.

McClelland, D. C. (1961). The Achieving Society. Princeton, NJ: Van Nostrand. http://dx.doi.org/10.1037/14359-000

McDonald N. and Goggins S. "Performance and Participation in Open Source Software on GitHub," in CHI – Extended Abstract, 2013, pp. 139–144.

Michael, J.B., Bossuyt B.J., and Snyder, B.B. (2002) "Metrics for measuring the effectiveness of software-testing tools," 13th International Symposium on Software Reliability Engineering,

2002.

Perry, D.E., Staudenmayer, N. A., Votta, L. G. (1994) People, organizations, and process improvement, Software, IEEE 11 (4) 36–45

Pmbok® Guide, “Project Management Body of Knowledge (PMBOK® Guide),” 4th Edition, Newtown Square, 2008.

Prechelt, L., Unger-Lamprecht, B., Philippsen, U., and Tichy, W. F. (2002) "Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance," in IEEE Transactions on Software Engineering, vol. 28, no. 6, pp. 595-606, June 2002, doi: 10.1109/TSE.2002.1010061.

Pressman, Roger S. (2005) Software engineering: a practitioner's approach. Palgrave macmillan.

Proynova, Rumyana & Paech, Barbara. (2013). Factors Influencing User Feedback on Predicted Satisfaction with Software Systems. 96-111. 10.1007/978-3-642-37422-7_7.

Ramaprasad, Arkalgud. (1983). On the Definition of Feedback. Behavioral Science. 28. 4 - 13. 10.1002/bs.3830280103.

Rajeev Kumar Gupta, Shivani Jain, Bharat Singh, and Sanjay Kumar Jha. (2019). Key Factors in Scaling up Agile Team in Matrix Organization. In Proceedings of the 12th Innovations on Software Engineering Conference (formerly known as India Software Engineering Conference) (ISEC'19). Association for Computing Machinery, New York, NY, USA, Article 23, 1–5. https://doi.org/10.1145/3299771.3299793

Rasheed, Anwar & Khan, Saif-ur-Rehman & Rasheed, Mazen & Munir, Yasin. (2015). The Impact of Feedback Orientation and the Effect of Satisfaction With Feedback on In-Role Job Performance. Human Resource Development Quarterly. 26. 10.1002/hrdq.21202.

Rosenberg, J. "Some misconceptions about lines of code," Proceedings Fourth International Software Metrics Symposium, 1997, pp. 137-142, doi: 10.1109/METRIC.1997.637174.

Saavedra, R., Ballejos, L.C., & Ale, M.A. (2013). Software Requirements Quality Evaluation: State of the art and research challenges.

Sach, R. and Petre, M. (2012) "Feedback: How does it impact software engineers?," (2012) 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE), 2012, pp. 129-131, doi: 10.1109/CHASE.2012.6223008.

Sach, R., Sharp H., and Petre, M. (2011) "Software Engineers' Perceptions of Factors in Motivation: The Work, People, Obstacles," 2011 International Symposium on Empirical Software Engineering and Measurement, 2011, pp. 368-371, doi: 10.1109/ESEM.2011.50.

Sach, R., Sharp, H., and Petre, M.(2011) “What makes software engineers go that extra mile?” 23rd Annual Psychology of Programming Interest Group 2011, 6-8 September 2011, York, UK

Sach, Rien (2013). The Impact of Feedback on the Motivation of Software Engineers, PhD

Shaw, M., and D. Garlan, (1995) “Formulations and Formalisms in Software Architecture,” Volume 1000—Lecture Notes in Computer Science, Springer-Verlag.

Singh, B. and Gautam, S., (2016) "The Impact of Software Development Process on Software Quality: A Review," 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN), 2016, pp. 666-672, doi: 10.1109/CICN.2016.137.

Stockman, S. G., Todd, A. R., & Robinson, G. A. (1990). A framework for software quality measurement. IEEE Journal on Selected Areas in Communications, 8(2), 224–233. https://doi.org/10.1109/49.46876

Succi, G., Benedicenti, L., & Vernazza, T. (2001). Analysis of the effects of software reuse on customer satisfaction in an RPG environment. IEEE Transactions on Software Engineering, 27(5), 473 –479

Sun, Hongyi & Ha, Waileung & Xie, Min & Huang, Jianglin. (2014). Modularity's impact on the quality and productivity of embedded software development: a case study in a Hong Kong company. Total Quality Management & Business Excellence. 26. 1-14. 10.1080/14783363.2014.920179.

Sutton, C. D., & Woodman, R. W. (1989). Pygmalion goes to work: The effects of supervisor expectations in a retail setting. Journal of Applied Psychology, 74(6), 943–950. https://doi.org/10.1037/0021-9010.74.6.943

Thongtanunam, P. and Hassan, A. E. (2020) “Review dynamics and their impact on software quality,” IEEE Transactions on Software Engineering.

Thorndike,E. L. (1927). The law of effect. American Journal of Psychology, 39, 2 t2-222.

Tsay, J., Dabbish, L., and Herbsleb, J. (2014). “Influence of Social and Technical Factors for Evaluating Contribution in GitHub,” in ICSE, 2014, pp. 356–366.

Unterkalmsteiner M., Gorschek T., Islam A.M., Cheng C.K., Permadi R.B., Feldt R. (2012). Evaluation and measurement of software process improvement a systematic literature review, Software Engineering, IEEE Transactions on 38 (2)398–424.

Urbach, Nils & Müller, Benjamin. (2011). The Updated DeLone and McLean Model of Information Systems Success. 10.1007/978-1-4419-6108-2_1.

Wang, Ying & Yu, Hai & Zhu, Zhi-liang & Zhang, Wei & Zhao, Yu-li. (2018). "Automatic Software Refactoring via Weighted Clustering in Method-Level Networks," in IEEE Transactions on Software Engineering, vol. 44, no. 3, pp. 202-236, 1 March 2018, doi: 10.1109/TSE.2017.2679752.

Wedyan, F. and Abufakher, S. (2020), Impact of design patterns on software quality: a systematic literature review. IET Softw., 14: 1-17. https://doi.org/10.1049/iet-sen.2018.5446

Weiner, I. B., et al. (2003) Handbook of Psychology Assessment Psychology. Pp. 123 (John

Wiley and Sons, 2003)

White, S., & Locke, E. (2000). Problems with the Pygmalion effect and some proposed solutions. Leadership Quarterly, 11, 389–415.

Zhang, Hongyu. (2009). An Investigation of the Relationships between Lines of Code and Defects. IEEE International Conference on Software Maintenance, ICSM. 274-283. 10.1109/ICSM.2009.5306304.

Zowghi D. and Gervasi, V. “On the interplay between consistency, completeness, and correctness in requirements evolution,” Information and Software technology, vol. 45, no. 14, pp. 993–1009, 2003.