The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM)
ISSN: 2602-3199

The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM)

Volume 1, Pages 411-419

ICONTES2017: International Conference on Technology, Engineering and Science

SOLUTION OF CONSTRAINTS THROUGH THINKING PROCESS IN SOFTWARE CODE REVIEWS

Yildiz Sahin
Kocaeli University

Pinar Yildiz Kumru
Kocaeli University

Ozgur Gun
Kocaeli University

Abstract: The high quality of software produced in software projects is vital for the success and life of institutions. Software defects can lead to reduced customer satisfaction, increased maintenance costs, and / or reduced productivity and utility. Software code review is one of the most important techniques used to improve software quality and reduce errors in software. Software code review is the process by which one or more people evaluate a code before writing a new version and putting it in a code repository.

The main purpose of the code is to prevent defects and to be able to understand the code at the same time as it is to carefully examine the code against deviations from the development standards. Another goal is to help people who write code to produce much better products in the future. In this study, it is aimed to investigate the reasons why the code review process cannot be implemented and to solve these reasons with the “Theory of Constraints Thinking Process”.

Keywords: Theory of constraints, thinking processes, software code review process

Introduction

Theory of Constraints

Theory of Constraints (TOC) was discussed and developed in the book published by Goldratt in 1984. TOC was first time; It was used for production planning and scheduling to increase profit taking into account market conditions. The latter was also used in marketing, sales and information systems. TOC was not widely used in profit-oriented organizations such as government and administrative services (Goldratt (1990a) [1], Goldratt (1990b) [2]).

Goldratt (1990c) [3], Klein and DeBruine (1995) [4] and Dettmer (1997) [5] consider the constraint theory that an organization consists of interconnected chain rings. From this perspective, a system is as strong as its weakest link. According to Goldratt, if you want to improve the performance of a system, you must first find its weakest rings, in other words, its constraints.

In the theory of constraints; Constraints affect the performance of the system negatively. Many classifications ca n be made about constraints.

Goldratt divided the constraints into two, internal and external. External constraints are those that are outside the organization and organization have little effect on their control. Internal constraints are divided into two groups, physically and politically. Physical constraints; hardware and staff. Political constraints include the organization-wide practice of breaking the system.

*Corresponding author: Yildiz Sahin-E-mail: yildiz.sahin@kocaeli.edu.tr

Ring and Perry (1985) [6] point out that there is a difference between constraints in the public and private sectors.

In this study, it was tried to gather information about the process performance of the code review processes of the projects carried out in a public organization in the military and civil projects in Research and Development activities. As a result of reviewing this information, it has been found that the biggest problems in the code review process of software projects are not to allocate sufficient resources (budget, time, labor) for code checking activities in project planning, not to operate the code review process of the project team and not to use a tool for code checking. In this study, "Thinking Process of the Theory of Constraints" was used to find out and solve the main cause of the problem.

Method

Constraints Management

A five-step model of Goldratt’s book aim to analyze and manage constraints has been identified. This model is one of the most important parts of the constraint theory.

1. Identify constraints
2. Get the greatest benefit possible from constraints
3. Design other activities / processes according to the properties of the constraint
4. Add resources to improve performance
5. Go back to the first step and start searching for the new constraint

The above model is a systematic approach developed to manage constraints. There are many tools and techniques to operate this model. The Drum-Buffer-Rope (drum-buffer-ip) scheduling technique indicates that the process follows a tapping sound when it produces a constraint piece. At the end of the process, the finished product buffer is used so that the final product shipment is not adversely affected if it cannot produce constrained parts. A rope is a signal that when a product is sold, the part entry must be done.

Thinking Process

Another technique in the management of constraints is the Thinking Process. According to Goldratt (1990a) [1], managers who deal with constraint management must make three decisions.

1. What will change in the system?
2. What will it turn into?
3. How will the change take place?

To address these questions, the thinking process presented the following five tools.

1. Current reality tree
2. Evaporating cloud
3. Future reality tree
4. Prerequisite tree
5. Transition tree

Current Reality Tree (CTR): The first step in implementing thinking processes is to list the undesirable effects and build the existing reality tree accordingly. CRT is designed to analyze the current state of a system and to better understand the problems, and to identify the basic problems with undesirable effects that reduce the performance of the system. CRT is a diagram showing cause-and-effect relationships between adverse effects and their consequences. The aim is to find the root cause of the problem. First the root cause is found and removed. Thus, unwanted effects disappear.

Evaporating Cloud (EC): The solution to remove the unwanted result is the tool in which the basic and preliminary requirements are defined, the clash between the solutions is made, and the injection is made to destroy the clash. This tool involves the handling of a single problem separately, the determination of conflicts and assumptions encountered, and the examination of a solution. The evaporative cloud method acts as an effective bridge, contributing to the removal of problems from the current state of affairs in the transition to the desired future state.

Future Reality Tree (FRT): A tool used to imagine, animate and predict the future. Future reality tree shows the causal relationship between the changes to be made in the existing system and the consequences that may occur. FRT is a what-if application. It tries to determine the benefits of the proposed change, the negative effects it will have, and how to remove these effects.

Prerequisite Tree (PRT): Provides a logical way to create secondary solution clusters that are required to come up from the top of all obstacles in front of the solution idea. The aim is to help define all of the intermediate steps needed to achieve a great goal. The development of the pre-requisite network defines local barriers, situations and omissions that prevent achievement of the desired outcomes and sets new goals and objectives that will enable these barriers and change to come from the superior.

Transition Tree: The transition tree is used to identify the activities required to achieve the purpose. It is a cause-and-effect chain designed to reveal step-by-step processes from the definition of an undesirable outcome to the completion of change.

Software Code Review Process

Software review process is an endorsed process that improves the quality of the software product and reduces software development life cycle time and costs.

Software reviews are recommended as a cost-saving quality assurance technique to improve the Software Development Process and are widely used in industrial applications.
Software reviews are defined as "techniques that are used to carefully examine software products against deviations from development standards and that do so without performing software".
The main goal of software review is to find errors during the Software Development Process. The goal of the software oversight is not only to find mistakes, to gather improvement suggestions, but also to help people who write code to produce better products in the future.
Software reviews can be classified based on the level of formalism in the oversight process or on the level of severity and flexibility.
Formal code reviews have a very attentive process structure, which requires support for advanced planning and corporate infrastructure.

Informal reviews are unstructured processes that are built on the desire to meet the needs of specific situations. Such reviews require less time, lower costs, no need for advanced planning and support for the institutional infrastructure.

The software code review typically consists of the following activities: providing the criteria for starting the review, setting the units to be reviewed, making an announcement to the people who will participate in the review (individual reviewers), sending bugs/defects to software developers, making related corrections, sending updated software units back to the reviewers for inspection, approval of all reviewers and closing reviews.

In recent years, many open source and commercial projects have used the code review process. In the following selected publications, information is given about the factors that affect the efficiency and effectiveness of the code review process, and the studies on the technical and non-technical benefits of reviewing.

Poerter et al. [7] investigated the factors that influence the effectiveness of the traditional phagan examination method. These factors are the code unit, auditor and project team, respectively. The most influential factors in this study are the code size and functionality and the viewer's experience.

Rigby and Bird [8, 9] published publications on cooperative code observing practices in open source projects. They compared the oversight processes in commercial projects and open source projects. Although there are differences between projects, there are similarities between many characteristics, such as the time interval between reviews, the number of defects and the number of reviewers.

Bachelli and Bird [10] conducted research on the aims and results of modern code reviews. Although the primary purpose of reviewing is to find errors in software, they have found that only a fraction of the findings / comments found in the review result are defects. They also argued that code reviews provides information sharing, team awareness and better solutions.

Baysal et al. [11] found that factors such as software size, component type, viewer experience / knowledge level, and experience of code writer were significant influences on code review.

Bosu et al. [12] investigated the factors that influence the effectiveness of code review. As a result of analyzing the code review results made on the projects realized in Microsoft company; a) The experienced software engineers who have worked in the institution for many years have more effective reviews, b) The effectiveness of the reviews is decreased by increasing the number of rows in the software components to be watched, c) The weaknesses in the file types or software components in the software system can be determined by calculating the code review efficiency.

Bosu, Carver et al. [13] studied the non-technical benefits of the review process. He also made suggestions on how to properly interpret comments / findings and how to better understand the software through which reviewers can see.

Sripada et al. [14] pointed out that as a result of a research conducted in a software engineering classroom of 200 people in India by second-year college students, increased coding skills, improved program understanding capabilities, increased knowledge of coding standards, and increased team communication.

Bernhart et al. [15], based on an industrial firm; evaluated the results of 114 surveys conducted within 18 months on a project which 8 software engineers worked. At the end of the evaluation; seeing that code review has positive effects on understanding the software code and making the software co-ownership.

Mantyla et al. [16] handled a total of 32 code reviews, 9 of which are industrial and 23 of which are students. It has 2 classes as error class. The first is functional and the second is the class of evolvability. It has been argued that finding the type of evolvability defects can contribute to the long life of software products. As for the classes of evolvability defects, structure, visual presentation, documentation types. As a result; pointed out that the code review process is a good tool for revealing the types of evolvability defects.

Findings

Usage of Thinking Process of Theory of Constraints in the Software Code Review Process

In this study, it was tried to gather information about the process performance of the code review processes of the projects carried out in a public organization in the military and civil projects in Research and Development activities. As a result of examining this information, it has been found that the biggest problems in the code review process of software projects are not to allocate sufficient resources (budget, time, labor force) for code review activities in project planning, not to operate the code review process and not to use a tool for code reviews. In this study, "Thinking Process of the Theory of Constraints" was used to find out and solve the main cause of the problem in software code reviews.

The first step in implementing the thinking processes is listing the unwanted effects, then creating the Current Reality Tree (CRT). CRT is a diagram showing the causal relationship between adverse effects and their consequences. The goal is to find root cause. When this cause is found and removed, the unwanted effects disappear. The main constraints encountered in the software code review process are; absence of support tools in code review activities, lack of definition of code review process, failure of project employees to implement processes, and lack of definition of performance measurement and evaluation process. Figure 1 The Current Reality Tree (CRT) reveals the root causes of the problems experienced in the code review process. Problems with the "*" sign are the main problems that lead to the problem.

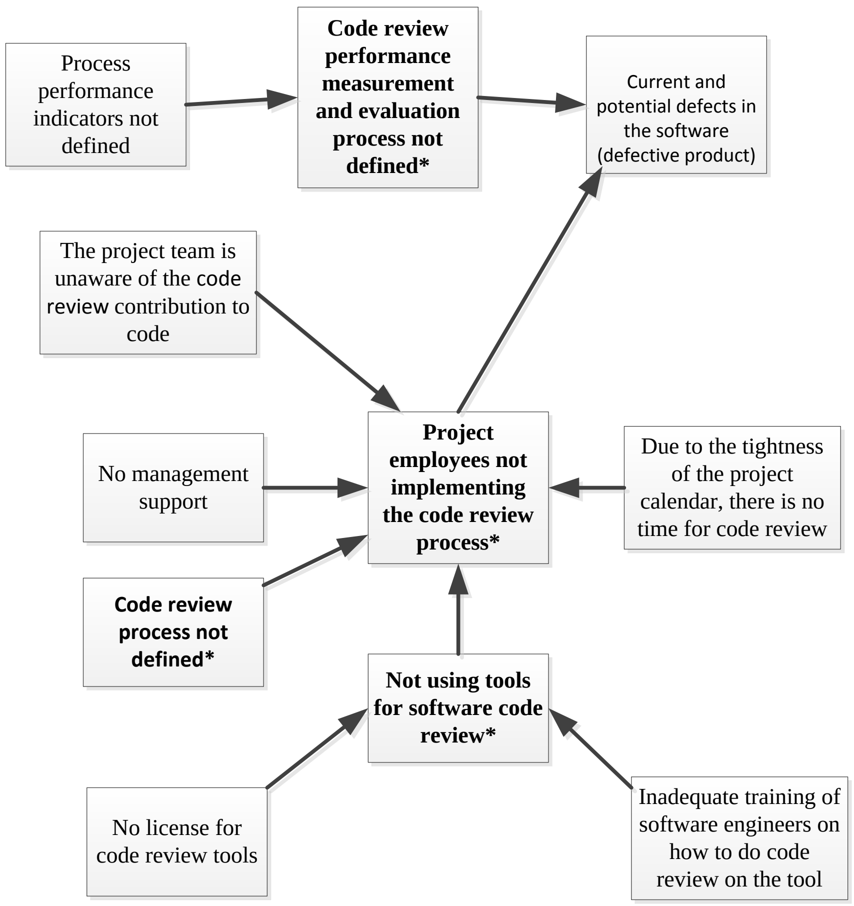
Process performance indicators not defined
Code review performance measurement and evaluation process not defined*
Current and potential defects in the software (defective product)
The project team is unaware of the code review contribution to code
Project employees not implementing the code review process*
No management support
Due to the tightness of the project calendar, there is no time for code review
Code review process not defined*
Not using tools for software code review*
No license for code review tools
Inadequate training of software engineers on how to do code review on the tool

Figure 1. Current reality tree (CRT)

The second step in the process of thinking is the creation of the Evaporating Cloud (EC). The evaporating cloud technique is used to reveal the solutions of the problems that present the root cause.

In this technique, after the conflict is revealed, injection(s) are recommended for destroying this conflict. Figure 2 The Evaporating Cloud shows one of the conflicts that can be encountered in the software code review process and the possible injections for the solution. The conflict here is whether project employees implement the code review process. Injections to remove this conflict are the addition of resources (labor, time and budget) required for code reviews in preparation of the project contract, and training of project staff about the code review process. By applying these injections, it is possible to remove the root cause that leads to the problems.

.

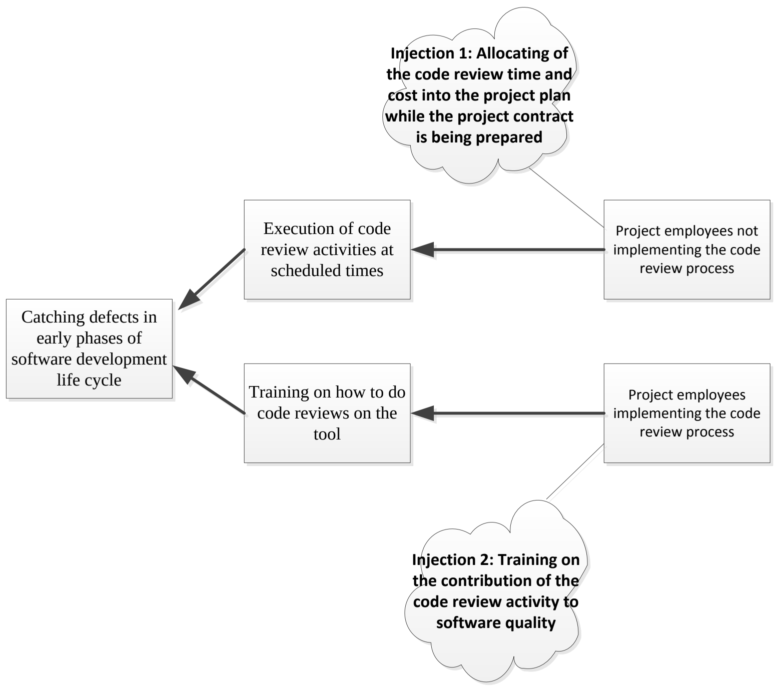
Injection 1: Allocating of the code review time and cost into the project plan while the project contract is being prepared
Execution of code review activities at scheduled times
Project employees not implementing the code review process
Catching defects in early phases of software development life cycle
Training on how to do code reviews on the tool
Project employees implementing the code review process
Injection 2: Training on the contribution of the code review activity to software quality

Figure 2. Evaporating cloud

The next step is the implementation of the Future Reality Tree (FRT). The intent of this phase is to confirm that a desired state will achieve the expected best results (Desired Effect - DE). Figure 3 The Future Reality Tree (FRT) shows the future reality tree of this work. In FRT, productivity increases (code, documentation development time, etc.) and cost of code development are reduced as project employees mentioned in Figure 2 Evaporative Cloud diagram begin to implement the code review process. By resolving the errors in the code, the customer will receive fewer error-intensive products. It improves the quality of product software without any mistakes, it provides customer satisfaction. Increased customer satisfaction enables the same client to win more projects. Thus the software firm can increase its capacity and profit.

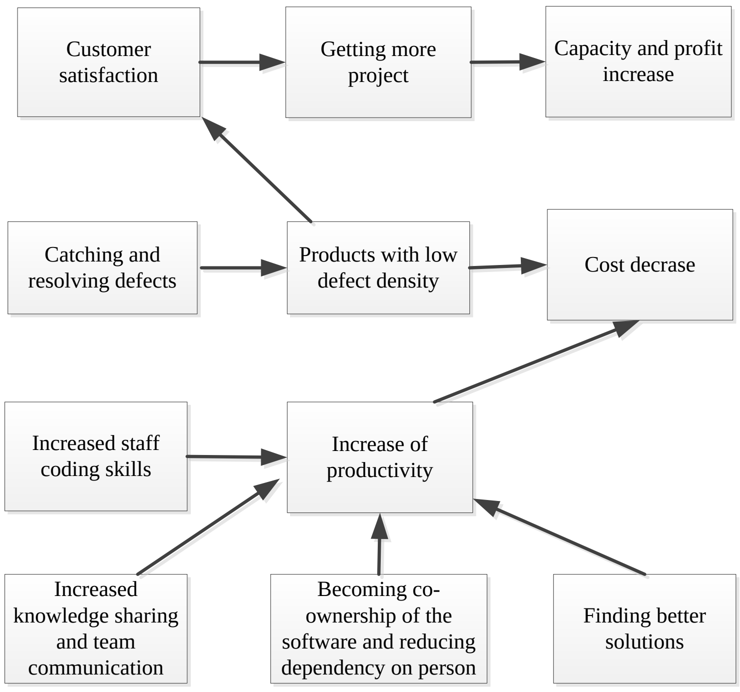
Customer satisfaction
Getting more project
Capacity and profit increase
Catching and resolving defects
Products with low defect density
Cost decrase
Increased staff coding skills
Increase of productivity
Increased knowledge sharing and team communication
Becoming co-ownership of the software and reducing dependency on person
Finding better solutions

Figure 3. Future reality tree (FRT)

The next step is to Prerequisite Trees (PRT) and Transition Trees to be able to pass on the changes and create solution clusters. With the Prerequisite Tree, the situation that prevents me from reaching the solution is defined. As seen in Figure 4 Prerequisite Trees, there are not enough resources (labor, time, and budget) to run the code reviews in the projects. If these resources are not placed in the project contract during the proposal preparation stage, delaying the project schedule will be inevitable since the execution of the code review activities will require the expenditure of unplanned source. In order to remove this obstacle, budget is allocated for code reviews at the project proposal stage, personnel are assigned to train employees. With the support of the management, it is encouraged to employees and code reviews is made in the projects. Thanks to the reporting capabilities of the code review tools, code review process performance is monitored periodically or instantly.

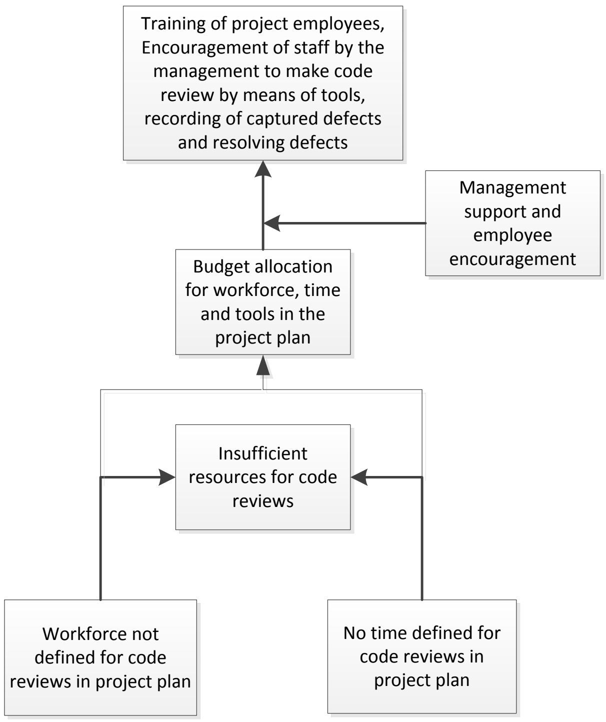
Training of project employees, Encouragement of staff by the management to make code review by means of tools, recording of captured defects and resolving defects
Management support and employee encouragement
Budget allocation for workforce, time and tools in the project plan
Insufficient resources for code reviews
Workforce not defined for code reviews in project plan
No time defined for code reviews in project plan

Figure 4. Prerequisite tree (PRT)

The transition tree is used in detail to define the activities planned to achieve the main objectives and to put them into practice.

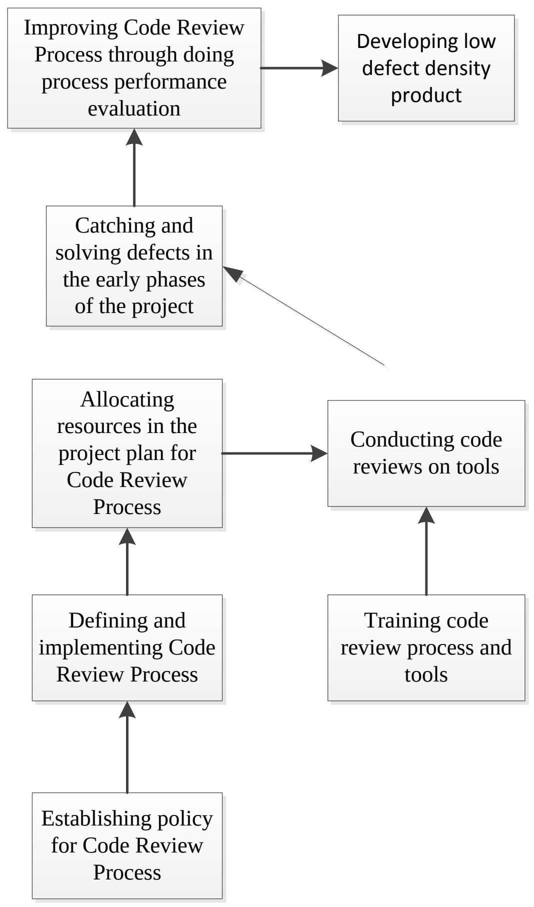
Improving Code Review Process through doing process performance evaluation
Developing low defect density product
Catching and solving defects in the early phases of the project
Allocating resources in the project plan for Code Review Process
Conducting code reviews on tools
Defining and implementing Code Review Process
Training code review process and tools
Establishing policy for Code Review Process

Figure 5. Transition tree

Figure 5 Transition Tree defines a policy by the enterprise management to produce error-low software, defines and implements the code review process, allocates resources for the code review process in the project plans, training on the code reviewing tool and monitoring the process performance and updating the process if necessary. Project process audits should be done, code review process performance should be measured and statistical process control methods should be applied. Points to be improved in processes should be identified and corrective actions should be carried out. Eliminating the bottlenecks in the code reviews will ensure delivery of high quality products to the customer, reducing both costs and error rates.

Results

One of the biggest challenges facing the software industry today is to drive the product market in a shorter time and at the same time to produce products with low error density. Rapidly pouring software products into products can result in insufficient time for product validation and validation, resulting in product quality being waived.

Software product quality will only be very costly and will cause the project to be delayed if only the finished product is tried to be tested and corrected. The cost of eliminating the faults detected at the end of the project is much higher than the cost of removing the faults detected at the beginning of the project. For this reason, it is very important that the code review process is executed correctly for every software component that is encoded. The reasons for not applying the code overhaul process are mainly; Project Managers and Software Team are not aware of the significance / importance of this process, and another reason is the lack of labor, time, budget and vehicle infrastructure resources required for this process.

Since the software code review process is only defined on paper, it is not enough to implement it on projects, and project support tools are needed to help implement these processes. Tools such as Bugzilla, Code Collaborator, and Understand are typical tools that can be used for the code review process.

The use of these tools in the organization depends on a strong governance policy, the identification of the code review process as a requirement of this policy, and the allocation of resources for code review tools.

The project management should follow up and evaluate the code review process. Software units above a certain level of error should not be included in the new version of the software product without errors.

It is believed that this study will provide a way for companies operating in the software industry to solve other difficulties (software unit test, component test, etc.) they encounter in the software development process.

References

Goldratt, E.M. (1990a)What is this thing called theory of constraints and how should it be implemented? Massachusetts:North River Press

Goldratt, E.M. (1990b). The haystack syndrome: Sifting information out of the data ocean. New York: North River Press

Goldratt, E.M. (1990c). What is this thing called the theory of constraints? New York: North River Press

Klein, D., & DeBruine, M. (1995). A thinking process for establishing management policies. Review of Business, 16(3), 31–37.

Dettmer, H.W. (1997). Goldratt’s theory of constraints: A systems approach to continuous improvement. Milwaukee: ASQC Quality Press

Ring, P.S., & Perry, J.L. (1985). Strategic management in public and private organizations: Implications of distinctive contexts and constraints. The Academy of Management Review, 10(2), 276–286.

Fagan, M. E. (1976). “Design and code inspections to reduce errors in program development,” IBM Systems Journal, vol. 15, no. 3, pp. 182 –211

Rigby P. C., & German D. M., & Storey M. A. (2008). “Open source software peer review practices: a case study of the apache server,” in Proceedings of the 30th international conference on Software engineering. ACM, pp. 541–550

Rigby P. C., & Bird C. (2013). “Convergent contemporary software peer review practices,” in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE, pp. 202–212.

Bacchelli A., & Bird C. (2013). “Expectations, outcomes, and challenges of modern code review,” in Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, pp. 712–721.

Baysal O., & Kononenko O., & Holmes R., & Godfrey M. W. (2013). “The influence of non-technical factors on code review,” in Reverse Engineering (WCRE), 2013 20th Working Conference on. IEEE, pp. 122–131

Bosu A., & Greiler M., & Bird C. (2015). Characteristics of Useful Code Reviews: An Empirical Study at Microsoft, 12th Working Conference on Mining Software Repositories

Bosu A., & Carver J.C., & Bird C., & Orbeck J., & Chockley C. (2017). Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft, IEEE Transactions On Software Engineering, Vol. 43, No. 1.

Sripada S., & Reddy Y.R., & Sureka A. (2015). In Support Of Peer Code Review And Inspection In An Undergraduate Software Engineering Course, 28th Conference On Software Engineering Education And Training

Bernhart M., & Grechenig T. (2013). On The Understanding Of Programs With Continuous Code Reviews, IEEE International Conference on Program Comprehension, San Francisco, USA

Mantyla M. V., & Lassenius C. (2009). What Types Of Defects Are Really Discovered In Code Reviews?, IEEE Transactions On Software Engineering, Vol. 35, NO. 3