UNIVERSITÉ DE MONTRÉAL

# SOFTWARE ENGINEERING TAXONOMY OF TEAM PROCESSES: A COMPLETENESS AND USEFULNESS VALIDATION

SEYED REZA MIRSALARI
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AOÛT 2013

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

# SOFTWARE ENGINEERING TAXONOMY OF TEAM PROCESSES: A COMPLETENESS AND USEFULNESS VALIDATION

présenté par : MIRSALARI Seyed Reza

en vue de l'obtention du diplôme de : maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. ANTONIOL Giuliano, Ph.D., président

M. ROBILLARD Pierre-N, Ph.D., membre et directeur de recherche

M. KHOMH Foutse, Ph.D., membre

# ACKNOWLEDGEMENT

I wish to thank, first and foremost, my supervisor Professor Pierre.N- Robillard, for his guidance, encouragement and continuous caring support during my graduate studies. I share the credit my work with my friend and colleague Mathieu Lavallée, for all the knowledge that I learned from him.

For this dissertation I would like to thank the jury members: Giuliano Antoniol, Pierre.N-Robillard and Foutse Khomh for their time, interest, and helpful comments.

Most of all, a special thank to my dear wife Elham, who has always been there with her love and encouragement during the past years and thank to my sweetie daughter Sonia, as well.

# RÉSUMÉ

**Objectif**: Délibérer les études sur le travail d'équipe en génie logiciel (GL) et trouver un moyen de mieux comprendre la dynamique des équipes dans le domaine. **Contexte**: Plusieurs cadres et modèles théoriques ont été présentés dans la littérature sur le travail en équipe: des cadres généraux, des cadres spécifiques de travail et des modèles pour des fonctions spécifiques. **Stratégie**: Étude sur les équipes et l'équipe de travail dans la littérature du génie logiciel. Étude sur la programmation en paire (PP) comme un échantillon du travail en équipe en GL. Développement d'un processus itératif revu systématique de la littérature (iSR), utilisé comme un outil de recherche pour la collecte et la synthèse des données de la littérature. **Méthodologie**: Une revue systématique de la littérature sur le travail en équipe en GL et sur la programmation en paires est effectuée. Les résultats de l'examen de la littérature sont utilisés pour une validation d'une taxonomie de GL des processus d'équipe. **Résultats**: La taxonomie de GL des processus d'équipe est un outil approprié pour la présentation des observations des pratiques d'équipe dans le domaine du GL. **Conclusion**: Selon la méthodologie et les articles soumis, la taxonomie de GL des processus d'équipe a été validée. Les variables contextuelles des pratiques de PP ont également été identifiées et Le processus ISR a été jugée utile pour les novices. **Application**: La taxonomie de GL est un outil qui peut être utilisé par les chercheurs ainsi que les gestionnaires de projets logiciels pour identifier et signaler tout type d'interactions observées dans les équipes et pour améliorer les performances de la gestion des équipes.

# ABSTRACT

**Objective:** To deliberate the studies on teamwork in Software Engineering (SE) and to find a way to better understand team dynamics in the SE domain. **Background:** Several theoretical frameworks and models have been presented in the teamwork literature: general frameworks, task specific frameworks and function-specific models. **Strategy:** Study on teams and team working in the software engineering literature. Study on pair programming as a sample practice of teamwork. Development of an iterative systematic literature review (iSR) process used as a research tool for gathering and synthesizing data from the literature. **Methodology:** A systematic literature review on team working in SE and on pair programming is performed. The literature review results are used for an evaluation of current team working practices (namely PP practices) and for the validation of the SE taxonomy of team processes. **Results:** The SE taxonomy of team processes is an appropriate tool for the report of observations in teamwork practices in the SE domain. **Conclusion:** According to the employed methodology and submitted articles, the software engineering taxonomy of team processes was validated. The contextual variables of PP practice were also identified. The iSR process was found to be useful for novices. **Application:** The SE taxonomy is a tool which can be used by researchers as well as software project managers for identifying and reporting any kind of observed teamwork interactions. Report and analysis of team activities could improve team management performance.

# TABLE OF CONTENTS

ACKNOWLEDGEMENT ..... III
RÉSUMÉ ..... IV
ABSTRACT ..... V
TABLE OF CONTENTS ..... VI
LIST OF TABLES ..... VIII
LIST OF FIGURES ..... IX
LIST OF ANNEXES ..... X
INTRODUCTION ..... 1

CHAPTER 1: LITERATURE REVIEW ..... 8
1.1. Team, Teamwork and Team members ..... 8
1.2. Teamwork in Software Engineering and Software Project Management ..... 9
1.3 Team Processes in Software Engineering ..... 9
1.4 iterative Systematic Review (iSR) ..... 10

CHAPTER 2: ARTICLE: SOFTWARE ENGINEERING TAXONOMY OF TEAM PROCESSES: A COMPLETENESS AND USEFULNESS EVALUATION ..... 13
2.1. Introduction ..... 13
2.2. The taxonomy ..... 14
2.3. Perspectives ..... 16
2.3.1. Management perspective (Hughes and Cotterell Resource) ..... 16
2.3.2 Empirical study perspective (Literature Review Resource) ..... 17
2.4 Methodology ..... 18
2.5 Mapping process ..... 19
2.5.1 Management perspective (Hughes and Cotterell Resource) ..... 19
2.5.2 Empirical study perspective (Literature Review Resource) ..... 19
2.6 Results ..... 20

2.6.1 Management Perspective (Hughes and Cotterell Resource) ..... 20
2.6.2 Empirical studies perspective (Literature Review Resource)..... 21
2.7 Out of scope instances..... 23
2.8 Discussions..... 24
2.8.1 Management perspective (Hughes and Cotterell Resource)..... 24
2.8.2 Empirical study perspective (Literature Review Resource) ..... 24
2.9 Threats to validity and reliability ..... 26
2.10 Conclusion..... 27
CHAPTER 3: GENERAL DISCUSSIONS..... 29
3.1 Introductory sub-researches ..... 29
3.2 Relation with the main research topic ..... 30
CHAPTER 4: CONCLUSION ..... 32
4.1 Teamwork in software engineering..... 32
4.2 Pair programming..... 33
4.3 Systematic literature review ..... 33
REFERENCES ..... 34
ANNEXES..... 42

# LIST OF TABLES

Table 1. Team and Member categorization of team performance characteristics ..... 8
Table 2. Identified teamwork activities and mapped episode(s) for the Hughes et al. resource.[1] ..... 21
Table 3. Examples of identified teamwork activities and mapped episode(s) for one paper selected in the literature review. ..... 22
Table 4. Examples of identified teamwork activities mapped to episodes. ..... 23
Table 5. Episodes appearance in retrieved activities (Resource: Literature) ..... 24
Table 6. Episodes seen together with the same activity. ..... 25
Table 7. Results from 4P point of view ..... 26
Table 8. Three definitions of pair programming according to the literature ..... 45
Table 9. Context variables and their status ..... 53
Table 10. List of the explicit tasks required for each of the review method: the traditional SLR method, the EBSE , method, and the proposed iterative systematic review method (iSR). ..... 75
Table 11. Partial results of a term impact analysis on a search string ..... 90

# LIST OF FIGURES

Figure 1. Research study plan ..... 7
Figure 2. Taxonomy ..... 15
Figure 3. The steps in project management where staffing concerns are important [1]. ..... 17
Figure 4. Search string. Document type (DT) was restricted to conference articles (ca), journal articles (ja) and articles in press (ip). The KY field refers to the subject, title and abstract of the article. The PN field refer to the publisher's name. The CV field refer to the controlled terms (i.e standard keywords) of the article. The LA field refer to the language of the full text. ..... 18
Figure 5. The processes (P0, P1, P2) illustrating teamwork activities discovery and episode mapping. ..... 19
Figure 6. Difference in information exchange and exchange duration in jelled PP teams, mentoring pairs and BB interactions. ..... 48
Figure 7. Categorization of the PP papers selected. ..... 56
Figure 8. Amount of pairing in the studies selected. Papers marked with an asterisk (*) contain only academic data. ..... 57
Figure 9. Pairing levels in academic vs. industrial studies. ..... 58
Figure 10. Example of total effort, pairing effort and the various purposes of pairing. ..... 59
Figure 11. Papers published in the last ten years on distributed PP. ..... 63
Figure 12. Artist's view of effort expanded during a typical literature review performed using the iSR approach. ..... 78

# LIST OF ANNEXES

ANNEXE 1 - THE ANATOMY OF THE PAIR: A MAPPING STUDY OF HOW PAIR PROGRAMMING IS DEFINED IN PRACTICE ..... 42
ANNEXE 2 - PERFORMING SYSTEMATIC LITERATURE REVIEWS WITH NOVICES: AN ITERATIVE APPROACH ..... 71

# INTRODUCTION

## Generalities

*"Simply throwing people together will not immediately enable them to work together as a team."*[1]

What are called "teams"? As Dyer wrote in 1984, "a team is a social entity which is composed by members with interconnected tasks and shared and valued common goals". Team members can be organized hierarchically, vertically or cross functionally and sometimes dispread geographically, but integrated around goals in common [2], [3].

Existence of complex and difficult tasks in organizations is one the main reasons to move toward team and team working research. A basic definition of teamwork which says *"people working together to achieve a specific goal which is beyond the capabilities of individuals solo work"* [4] seems to imply that setting a goal for a team is the primitive and adequate condition for team success. However, even working with a team full of technical experts with all possible resources does not guarantee success. Having a favorable outcome also depends on other parameters. The team processes that team members employ in order to accomplish the work and achieve team objectives is also an essential factor for team success.

Several theoretical frameworks and models have been presented in the general teamwork literature [2]. This variety of teamwork frameworks shows the dynamics and importance of team working in today's industries. These models cover a spectrum of all known kinds of teamwork activities, from general frameworks [5] to task specific frameworks, [6] to function-specific models [7].

All of these theoretical models have some basic core concepts and general structures in common. The I-P-O conceptual model is one of these. It is a popular framework which looks at a teamwork activity as a series of processing elements mediated by Inputs and Outputs. Information flows through the I-P-O cycles and I-P-O network accordingly, based on the rules or decision points [4].

The I-P-O framework contains episodes to illustrate the period of times that team members perform their various tasks. The episodes are temporal cycles of goal-directed activity [4]. Several episodes can be combined together and run sequentially and simultaneously in order to configure an I-P-O framework to demonstrate the team performance trajectory. Episodes

specifically are “distinguishable periods of time over which performance occurs and feedback is available. [8] ”

A generic team process taxonomy based on the I-P-O framework defined by Marks et al. [4] has been adapted to the software engineering (SE) domain by Robillard & Lavallée [9]. They claim that in order to adapt the original Marks et al.'s taxonomy to the SE domain, “it’s necessary to consider the ‘taskwork’ episodes in addition to the ‘teamwork’ observations. Taskwork refers to what the team is doing, while teamwork refers to how they are doing it with each other” [9].

The feasibility of observing and reporting the software engineering teamwork activities using a specific vocabulary is the main subject of this dissertation. During this project, the validation of the SE taxonomy of team processes as a tool for reporting the teamwork observations has been deliberated. This research resulted in a scientific paper submitted to the IET Software journal. The paper is presented in chapter 2 of this dissertation.

Nowadays, teamwork has an essential role in SE domain. A practical instance of teamwork in software development is “pair programming” (PP). Since the date Kent Beck originally defined the PP practice, the arguments on its pros and cons are still being addressed. Pair programming, by definition, is a practice in which two programmers work together at one computer. The “process” used by teams members in a pair programming session is an essential factor for team success. Despite the many researches performed in the pair programming domain, there is still no clear answer as to what are the (dis)advantages of PP. Our study on pair programming resulted in a paper submitted to the IST journal. This paper is included in Annex 1 of this dissertation. In this research, a literature review on pair programming was performed and a set of deterministic context variables in the pair programming domain were identified.

For the study of pair programming, a *systematic literature review* process was used. The review retrieved all reported types of PP experiences performed in industrial environments. Literature review is usually the first step of any research project. Since the science is being done by researchers located all around the world, the work of each researcher should be continued by others. Accordingly, a researcher can employ a literature review (LR) process in order to perform a mapping study of the current and previously developed body of knowledge. The Systematic Literature Review process is a more concentrated LR method which gathers and evaluates the available evidence pertaining to a focused domain [10]. Our study on systematic literature review resulted in a paper submitted to the IST journal. This paper is included in Annex 2 of this

dissertation. In this research, the "iterative systematic literature review" was validated and its efficiency was demonstrated using the results of recent research on PP.

## Research problem

The lack of a unique, complete and useful vocabulary of teamwork in the SE domain causes the researchers to explain their observations and findings in various ways without considering a standard and uniform framework. One set of teamwork activities can be observed by two different researchers and be described in two different forms. These differences can lead to two different understandings of the same phenomena.

Defining a taxonomy for SE team processes has many advantages. It helps categorize the team activities. It can help project managers observe their teamwork episodes and easily detect the weakness and strength points of their team interactions. Observing team dynamics is a prerequisite for dealing with team effectiveness, but it is not sufficient. A project manager as well as a researcher needs a vocabulary to describe and report the observed dynamics. A taxonomy gives a complete set of terms and expressions to enable the users to express all the observed team activities. By a complete and useful vocabulary (taxonomy), a set of teamwork activities can be described succinctly and understandably for every audience who is familiar with the literature and the vocabulary.

## Roadmap: Objectives and Methodology

Compiling the teamwork studies and finding a way to better understand the team dynamics in the SE domain is the main objective of this dissertation. Team working in general has already been addressed in the literature and there is a valuable body of knowledge describing it from various perspectives. Nowadays, software development has become a full teamwork process. Teams are present in all phases of software development; from user needs gathering to design, code, test and maintenance. Several programming styles have been introduced which implies the existence and importance of teamwork in SE domain [11].

As I found, the unique vocabulary which so far has been defined in software engineering domain is the "software engineering taxonomy of teamwork processes". This taxonomy has been adapted to the software engineering (SE) domain by Robillard & Lavallée[9] originally from a generic team process taxonomy based on the I-P-O framework defined by Marks et al. [4].

As the main research goal presented in first journal article, we aimed to validate the appropriateness of the taxonomy for software development teams through evaluating its completeness and usefulness.

To achieve the research goal, a mapping process between teamwork activities identified in the literature with episodes defined in the software engineering taxonomy was performed. For this purpose;

- the standard episodes were extracted from the defined taxonomy,
- the selected literature was reviewed and the teamwork activities were identified,
- the identified activities were mapped to the relevant episodes of the taxonomy.

The validated results show that the software engineering taxonomy of teamwork episodes is a useful tool which can be employed by researchers and software project managers for identifying and reporting all observed teamwork interactions.

The second paper used pair programming as a sample of teamwork in the SE domain since pair programming is practiced in various styles and several contextual factors. All our research activities in the PP domain resulted in an article titled: “The anatomy of a pair: A mapping study of how pair programming is defined in practice”. This article, presented in the annex section of this dissertation, determines what context details are needed to properly identify the various definitions of PP and also what definitions of PP are used in studies performed in the field.

Our problems during PP researches as a practical sample of team working in SE domain leaded us to employ a tool: Systematic Literature Review. The result of this research has been presented in an article titled: “Performing Systematic Literature Reviews with Novices: An Iterative Approach”, which is available in the annex section. In this article we try to determine what difficulties and risks to address when performing systematic literature reviews with novices and how these risks can be mitigated.

## Original contributions

I divided this section into three parts, corresponding to the three articles, and each one presents the specific original contributions performed by me in order to achieve the pre-defined research objectives.

*A. "Software Engineering Taxonomy of team processes: A Completeness and Usefulness Evaluation"* [1] *– Chapter 2*

a. Strategy design and RQs definition: I designed the initial research methodology along with the research questions. I reviewed the research subject from two different perspectives. For each perspective, I considered a reference.
b. Data gathering: I selected a related set of articles and texts from the literature. I compiled all selected resources in order to fulfill the pre-defined data tables.
c. Performing the methodology; the mapping process: I defined three distinct sub processes. First, as described earlier, I selected the related teamwork articles and texts, then I retrieved the teamwork activities from each article and finally I mapped the retrieved activities to the identified teamwork episodes. The results were stored in data tables.
d. Results analysis: I analyzed all collected data and results in order to approach the research objectives. From two view points, I deliberated all identified teamwork activities and mapped episodes. I presented also the out of scope data.
e. Final concluding: The results were discussed. I identified the most and the least repeated episodes and also the combination of episodes on describing a teamwork activity. I reviewed the results from 4P (process, product, people and project) perspectives and the highlight points were identified and discussed. At the end, a validity check process was performed. As conclusion, I answered the research questions and addressed the completeness and usefulness of SE taxonomy of team processes.
f. Writing the paper: As the first author, I wrote the whole body of article. The co-authors helped me to revise the paper's structure and contents.

*B. "The anatomy of pair: A mapping study of how pair programming is defined in practice"* [2] *– Annex 1*

a. Research questions: Because of the nature of this work, I initially started to research with no specific research question. I used an iterative method to demonstrate the results of mapping study. I performed a mapping study to find the

[1] This article was submitted to IET Software journal on March 25, 2013.

[2] This article was submitted to TSE journal.

state of the art of pair programming and all the styles and context variables reported in the literature.

b. Information gathering: A search string (conceptual plan) was designed and performed on most related resources, i.e. Compendex, Inspec, ACM, and IEEE Xplore databases. The search string "pair programming" returned a lot of titles. Duplicates and noises were removed from the selection to find and gather the most related amount of articles.

c. Review the selected articles: During the review, we felt the need to determine the precise definition of PP used, which forms the basis of our research's final questions. Therefore we reviewed all the selected articles and analyzed them in order to specify the contexts of research i.e. academic or industrial, determine the type of research in terms of which definition is used to implement the PP experience, metrics used in each research and the outcomes.

d. Analyzing data tables and conclusion: we tried to find the amount of pairing in the selected articles. We also categorized the articles into two basic groups: industrial and experimental works and the recent group were categorized into lab experiments and field observations.

e. Continuous revision of the paper structure: in this article, I participated as the second author. Mostly, the body of article was written by first author and I helped him to revise the paper's structure continuously and alternatively.

*C. "Performing Systematic Literature Reviews with Novices: An Iterative Approach"* [3] *– Annex 2*

a. Data gathering for a case study: In order to validate the iterative systematic literature review (iSR), a review was conducted during the summer of 2012. The objective of this review was to perform a synthesis of the various definitions of pair programming practices reported in software engineering studies as a sample for iSR process validation. I was familiar with traditional literature reviews, but not with the iSR approach. I also was a domain novice, since I never participated in Pair Programming sessions, and had never researched the domain before. Therefore, I gathered the necessary data for pair programming practices as one of the cases

[3] This article was submitted to IST journal on March 14, 2013.

which studied during this research in order to demonstrate the efficiency of newly improved method (iSR).

## Organization of the dissertation

The next chapter presents a literature review. Chapter 2 contains the manuscript of the first article submitted to the IET journal, as described above. Then, Chapter 3 presents a general discussion, while Chapter 4 presents the conclusions, recommendations and future prospects of the research project. Two appendices are added to the end of the dissertation. Annex1 contains the second article which has been prepared on pair programming topic and Annex2 provides third article on iterative systematic literature review.

Figure 1 illustrates the plan which according to that our studies performed during the research period. It shows a summary of the structure of the areas that we touched in order to approaching the research objectives.

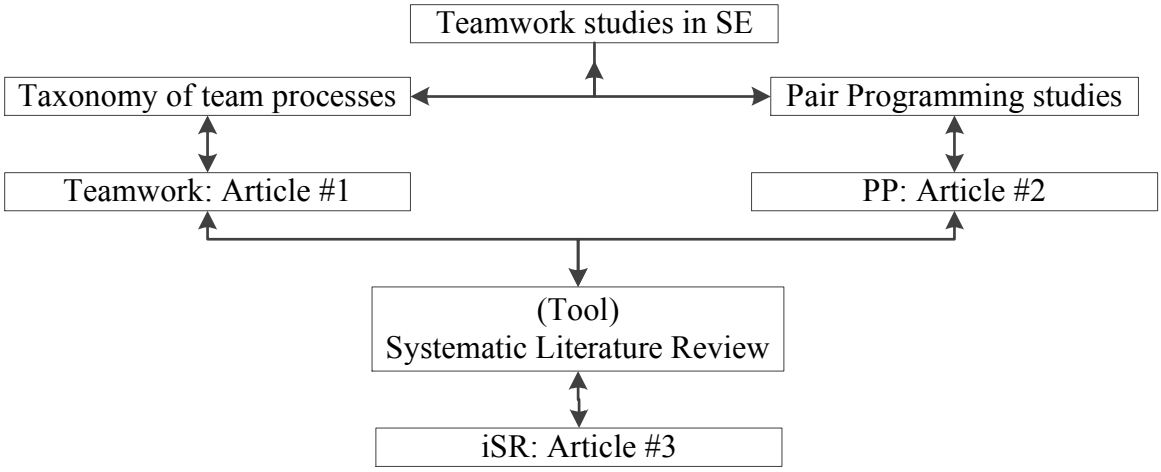


Figure 1. Research study plan

As it can be seen, our original objectives is finding a way to better study and understand the teamwork processes in software engineering. The iSR has been employed as a tool in order to collect and summarize the results gained from two higher level studies: pair programming and taxonomy of teamwork processes.

# CHAPTER 1:
# LITERATURE REVIEW

## 1.1. Team, Teamwork and Team members

Studies on scientific management date back to the 19th century, when it was addressed by Fredrick Winslow Taylor [12]. Organizational behavior (OB) as a part of scientific management of teams was defined and developed by researchers on the basis of Taylor's initial theories [13].

Taylor identified three basic objectives for the team: select the best team members, instruct them in the best methods, and offer higher wages to the best workers [13].

Although these characteristics are the minimum requirements, they do not give a comprehensive definition of team and teamwork. Katzenbach and Smith in 1993 [14] presented a more precise and comprehensible definition of the team as the following:

*"A team is a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable."* [14]

According to this definition, teams can be viewed from multiple aspects. The development of the team and of members' sense of belonging grows with time. Five stages of team development are suggested by Tuckman and Jensen [15]: Forming, Storming, Norming, Performing and Adjourning. In addition to these stages, it is necessary to distinguish effective teams from ineffective teams. In [16], twelve characteristics of an effective team are presented. These characteristics can be grouped in the two categories shown in Table 1. The "Team" category contains the characteristics related to the team's atmosphere, while the "Member" category is related to characteristics of the team members.

Table 1. Team and Member categorization of team performance characteristics

| | CHARACTERISTICS | CATEGORY |
|---|---|---|
| 1 | Informality | Team |
| 2 | Civilized Disagreement | Team |
| 3 | Clear Purpose | Team |
| 4 | Open Communication and Trust | Team |
| 5 | External Relations | Team |

| | | |
|---|---|---|
| 6 | Self-Assessment | Team |
| 7 | Style Diversity | Team |
| 8 | Participation | Member |
| 9 | Listening | Member |
| 10 | Consensus decision | Member |
| 11 | Clear Roles and work Assignments | Member |
| 12 | Shared Leadership | Member |

## 1.2. Teamwork in Software Engineering and Software Project Management

The literature in the field shows that Teamwork is a highlighted topic addressed in multiple references in the Software Engineering (SE) domain. In [17] p. 112, insufficient communication and teamwork has been considered as one of the obstacles in building a successful software system. In [18] (chapter 10), managing and leading are considered as two distinct activities: A competent project manager has been introduced as a person who is both a good manager and a good leader, or someone who finds ways to compensate for his or her weaknesses.

Hughes and Cotterell [1] discuss in detail software project management from various perspectives; effort estimation, risk management, activity planning, team management, teamwork, etc. They present a chapter on "Managing people and organizing teams" from a project manager point of view. They present a stepwise framework for software project planning and also highlight the steps where staffing concerns are important. They also identify some sub-steps for each of them and describe how to manage and perform these steps. In the current dissertation, this reference has been considered as one of the sources used to retrieve the teamwork activities from the project manager perspective.

## 1.3 Team Processes in Software Engineering

In order to report accurately the team interactions, a vocabulary needs to be defined. This vocabulary helps in the identification and diagnosis of team interactions. The advantage of this vocabulary is *"that it enables team process activities to be measured without reference to specific concepts related to perspectives other than that of the team process."* [9]. The work of Robillard and Lavallée [9] presents a vocabulary for the SE domain. They viewed a software product from the team process perspective. Their work uses a taxonomy of team processes defined in the psychology and business management fields [4] and adapt it to SE field. Their

work presents a new software engineering taxonomy of team processes with nine episodes which can be applied to report teamwork activities in the SE domain.

Ton That et al. [19] employed this taxonomy to show that their data collection method can be used to observe team process patterns. Their paper reached the conclusion that the quality of the SE taxonomy of team processes is sufficient.

The taxonomy defined by Robillard et al. [9] has been utilized in Ton That's study [19], but the validity of the taxonomy still needs to be evaluated some more. A validity evaluation on completeness and usefulness is required to show that the taxonomy is comprehensive enough to describe every teamwork activities in the SE domain. In the current project, I try to employ an efficient methodology to evaluate the appropriateness of the software engineering taxonomy of teamwork processes.

In addition, the Ton That [19] study uses only one case study to demonstrate that their data collection method is useful enough to observe patterns of team processes. However, in the current project, we have focused on the validity evaluation of the SE taxonomy of team processes using a larger number of teamwork reports and texts. This evaluation was performed based on two perspectives: the empirical study perspective and the management perspective. The results then show that the SE taxonomy is complete and useful enough to describe the teamwork activities.

## 1.4 iterative Systematic Review (iSR)

The traditional systematic literature review is a cascading review process which is built to answer specific research questions based on the conclusions of the high quality studies in a targeted domain. Biolchini et al. [10] presented the SLR process for software engineering as the five following steps:

1) Question formulation
2) Source selection
3) Studies selection
4) Information extraction
5) Results summarization

Another review method, evidence based software engineering (EBSE), also helps software engineers to review the literature. However, EBSE does not focus on answering questions, but on decision-making, and ensures that decisions are based on reported evidence. EBSE is adapted

from Evidence Based Medicine [20] for the software engineering domain. Kitchenham, Dyba and Jorgensen [21] showed that EBSE might have a variety of advantages for software teams and stakeholders. They identified the five main steps of the EBSE method according to the steps used in evidence-based medicine as below:

1) Converting the need for information into an answerable question.
2) Tracking down the best evidence with which to answer that question.
3) Critically appraising that evidence for its validity (closeness to the truth), impact (size of the effect), and applicability (usefulness in software development practice).
4) Integrating the critical appraisal with the software engineering expertise and with the stakeholders' values and circumstances.
5) Evaluating the effectiveness and efficiency in executing Steps 1-4 and seeking ways to improve them both for next time.

The iSR approach presented by Lavallée et al., appended to the current dissertation, has been based on the two above mentioned methods and has been derived from the practical experiences from four case studies. Its objective is to present an iterative approach to systematic literature reviews to address some of the problems faced by novice reviewers.

In the iSR approach, the literature review process is segmented in five major phases. Accordingly, each phase can be divided into several iterations. Eight tasks have been identified for planning the review process and the tutorial sessions in order to transfer the technical knowledge to the novice reviewers during the iSR process.

The iterations may involve each of the eight following tasks at various levels of effort and in various sequences.

1) Planning and training.
2) Question formulation.
3) Search strategy.
4) Selection process (relevance quality).
5) Strength of the evidence (research quality).
6) Analysis.
7) Synthesis.
8) Process monitoring.

Lavallée et al. demonstrated that an iterative approach of systematic literature review can be beneficial when the reviewers are domain novices. As the literature review requires a large amount of domain knowledge, it is recommended to benefit from domain experts. When domain experts are not available and/or the subject under review is too novel, unknown, or vague, finding domain experts can become a bottleneck for the research project. During the iSR, as the process advances, the reviewers' knowledge and their perception of the domain evolve, the research questions may need adjustments, the selection process may need adjustments, the data tables may require revisions, and the synthesis conclusion may need to be redesigned.

To achieve our goal in the current research project, we need to look at the teamwork activities from an empirical studies perspective. For this purpose, we should retrieve the team work activities that have been reported in the literature in the software engineering domain. So a literature review process was employed as a tool to select the related articles, and collect the needed data from them. We employed an iterative approach of systematic literature review, since we need the keywords to form and organize the search string, include related reports and exclude noises, constitute the data tables and make the research subjects more obvious and crystallized.

# CHAPTER 2:
# ARTICLE: SOFTWARE ENGINEERING TAXONOMY OF TEAM PROCESSES: A COMPLETENESS AND USEFULNESS EVALUATION

*Authors: Reza Mirsalari, Mathieu Lavallée, Pierre N. Robillard.*
*This article was submitted to IET Software journal on March 25, 2013*

## 2.1. Introduction

Research on teams goes back decades ago when investigators observed “group dynamics” in the social psychology field [22]. A recent literature review presented more than 130 theoretical models and frameworks of team performance [2].

One popular conceptual model is the I-P-O framework, which looks at a teamwork activity as a series of Input, Process and Output elements. Marks et al. [4] defined team processes as *“members’ interdependent acts that convert inputs to outcomes through cognitive, verbal, and behavioral activities directed toward organizing taskwork to achieve collective goals”* [4]. Marks et al. also presented a time-based team process model which can be applied to many team types.

The generic team process taxonomy of Marks et al. has been adapted to the software engineering (SE) domain by Robillard & Lavallée [9], which we will refer to as the Software Engineering Team Taxonomy (SETTaxo). They claim that in order to adapt the original Marks et al.'s taxonomy to the SE domain, it’s necessary to consider the ‘taskwork’ in addition to the ‘teamwork’ observations. Taskwork refers to what the team is doing, while teamwork refers to how they are doing it with each other [9]. An atomic element of a teamwork process is called an ‘episode’, which is a single time-based occurrence of an I-P-O cycle. Episodes may form a chain; the outcome of one episode can become the input for the following one [9]. In order to (1) observe and (2) report the teamwork activities, they first should be properly identified. The SETTaxo describes a set of "standard" episodes. The defined taxonomy containing standard

episodes should be comprehensive, which means that all teamwork or taskwork activities are described by episodes and no episodes remain unused.

Defining a taxonomy for SE team processes has many advantages. It helps to categorize the team activities and to detect the weaknesses and strength points of team interactions. Observing the team dynamics is a prerequisite for dealing with team effectiveness and a vocabulary is needed to describe and report the observed dynamics. This vocabulary is defined within the SETTaxo.

The aim of this paper is to validate the appropriateness of the episode taxonomy for software development teams (SETTaxo) through evaluating its completeness and usefulness.

The SETTaxo and its components are presented in section 2. Section 3 describes the perspectives which SETTaxo has been observed from. Section 4 presents methodology used to identify the activities from the selected studies and to map them to relevant episodes. The mapping process is described in section 5. Section 6 presents the results of the mapping process. Section 7 presents some out of the scope examples which define the limits of the taxonomy. Section 8 discusses the results and section. Section 9 presents the threats to validity and section 10 presents the conclusions.

## 2.2. The taxonomy

To perform the mapping between activities and episodes, it is necessary to have a reference model, as the workflow illustrated in Figure 5 describes. The SETTaxo reference model contains definitions of teamwork episodes, and some examples to clarify common issues with the activity/episode mapping [9]. This article presents an adaptation of the Marks et al.'s [4] taxonomy to the SE domain which should basically cover and describe all teamwork activities. The structure of the taxonomy and its episodes is listed in Figure 2.

The following presents the content of the SETTaxo and the definition of the episodes. An "episode" is an element or a set of elements of work which forms a part of a connected team process. It has a start and end point in time and a specific meaning. It is actually an atomic instance of an I-P-O cycle.

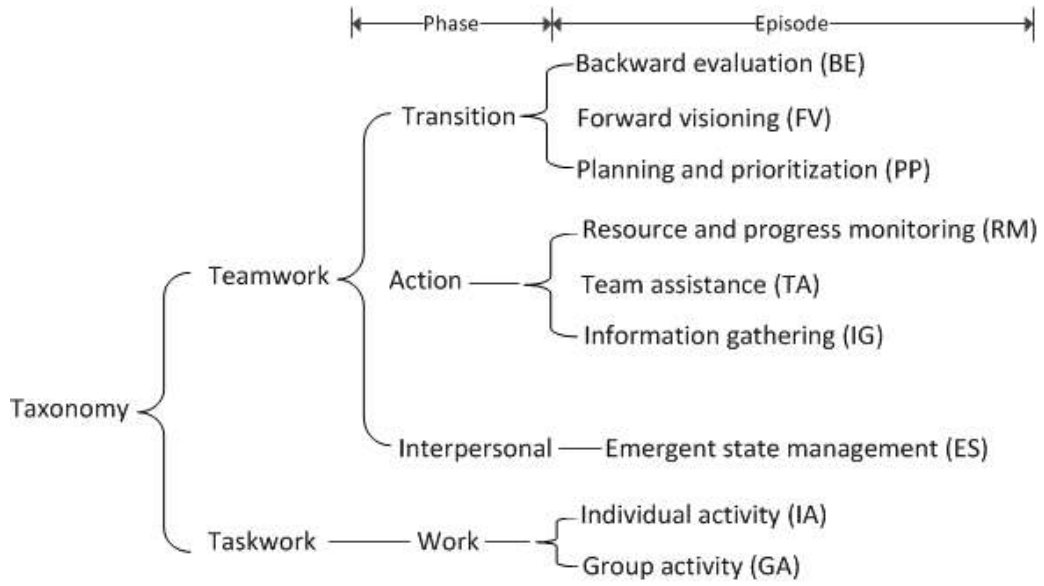


Figure 2. Taxonomy

The taxonomy includes four phases; transition, action, interpersonal and work phases. Each phase shows the status of team processes at a given time. "Transition" phase episodes are periods of times when team members are reviewing their past activities or trying to set a series of long/short term objectives and goals for the team. "Action" phase episodes are periods of time when team members interact between each other in order to achieve their goals. "Interpersonal" phase episodes are episodes dealing with personal motivation and emotions. The objective of interpersonal phase episodes is to increase the positive team feeling and team effectiveness and also diminish negative states like interpersonal conflicts [9]. According to the nature of work in SE teams, in addition to teamwork, taskwork also should be considered in order to describe all teamwork activities. *"Taskwork embodies a single phase, the 'work' phase, which describes the approach taken to perform the work."* [9]

In the following, a brief description for each episode inside the phases will be presented.

- **Transition Phase.**

i. **Backward Evaluation episode (BE):** During this I-P-O cycle, teammates evaluate the previously generated process artifacts [9].

ii. **Forward visioning episode (FV):** Designing a long term plan for project or retrieving user needs can be examples of FV episodes.

iii. **Planning and Prioritization episode (PP):** "This involves decision making about how team members will go about achieving their missions, discussion of expectations, rely of task-

related information, prioritization, role assignment, and the communication of plans to all team members" [4].

- **Action Phase.**

i. **Resource and Progress Monitoring episode (RM):** The episode "RM involves providing feedback to the team on its goal accomplishment status so that members can determine their progress and their likelihood of success within a given period of time" [4].

ii. **Team Assistance episode (TA):** During this episode, team members help each other to find an optimum solution for a requested problem.

iii. **Information Gathering episode (IG):** During this episode, team members try to obtain needed information from sources outside the team borders.

- **Interpersonal Phase.**

i. **Emergent state management episode (ES):** During this episode, team members try to prevent, manage or resolve emotional states inside team.

- **Work Phase.**

i. **Individual Activity episode (IA):** The nature of software development process implies that some tasks can (or should) be performed solely toward team goals. When a team member is performing a task alone, he is actually working in an IA episode.

ii. **Group Activity episode (GA):** This episode happens when team members are performing task together. Pair programming sessions are examples of GA episodes.

## 2.3. Perspectives

The teamwork in SE has been observed from two perspectives; first from the software project management perspective and second from the reported empirical experience perspective. Consequently, the information required for mapping process have been gathered from two major channels: first from Hughes and Cotterell [1], which gives a snapshot of teamwork activities from a "project management" perspective, and second from teamwork experiences which have been reported in related articles among the literature.

To collect the required information, the two resources were studied to extract the relevant teamwork activities. The two sub-sections below show the methods used for information gathering.

### 2.3.1. Management perspective (Hughes and Cotterell Resource)

Hughes and Cotterell posed the following question: *"Why is software project management important? Is a software project really that different from that of other types of projects?"* [1]Their answer refers to the key ideas of planning, monitoring and control of software projects. Answer to this question is important because it shows that the nature of teamwork in the SE domain is different from teamwork in other domains.

For this purpose, the work of Hughes and Cotterell [1] was chosen because its authors have discussed in detail about software project management from various perspectives; effort estimation, risk management, activity planning, team management, team working, etc. In chapter 11, the authors present "Managing people and organizing teams" from a project manager point of view, which is related to our taxonomy validation.

They have presented a stepwise framework for software project planning and also highlighted the steps where staffing concerns are important, as shown in Figure 3 [1]. They also identified some sub-steps for each of them and described how to manage and perform these steps. This research uses these sub-steps as a list of teamwork activities, and then tries to associate them to relevant episodes.

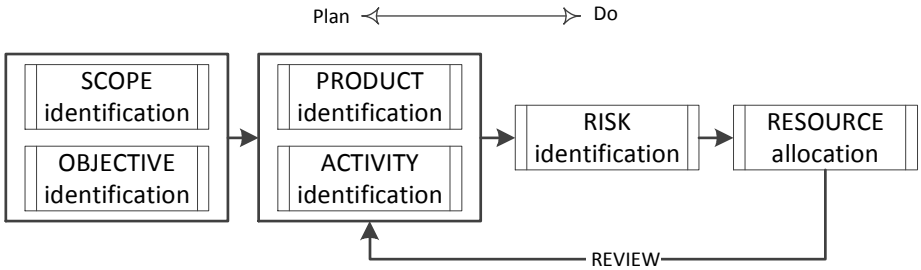


Figure 3. The steps in project management where staffing concerns are important [1].

## 2.3.2 Empirical study perspective (Literature Review Resource)

Scientific literature reports on many empirical studies on teamwork activities. The Engineering Village platform was used to select the primary materials for this study. The search was limited to the Compendex/Inspec databases of the Engineering Village because they include the most relevant journals and conferences in the SE domain. The search string ["Team" and "Software"] returned more than 14000 records. Restrictions were applied to the search string in order to retain a manageable set of papers, as shown in Figure 4. This more specific search string yielded 848 papers. Duplicates, proceedings titles, advertising tracts, research proposals, keynote presentations, and non-English papers were removed from the selection.

Finally, this mapping study is based on the first 19 most relevant papers, according to the Engineering Village platform.

((team) WN KY)
**AND**
((software) WN KY)
**AND**
({english} WN LA)
**AND**
(({ca} OR {ja} OR {ip}) WN DT)
**AND**
(({ieee} OR {ieee computer society} OR {springer verlag} OR {springer-verlag} OR {ieee comput. soc} OR {elsevier} OR {acm} OR {elsevier ltd} OR {john wiley and sons ltd} OR {elsevier science b.v.} OR {springer} OR {elsevier inc.} OR {springer netherlands} OR {john wiley sons ltd.} OR {elsevier science ltd.}) WN PN)
**AND**
(({john wiley sons ltd.} OR {elsevier science ltd} OR {ieee comput. soc.} OR {springer - verlag}) WN PN)
**NOT**
(({internet} OR {user interfaces} OR {mobile robots} OR {multi-robot systems} OR {mathematical models} OR {teaching} OR {virtual reality} OR {computer simulation} OR {artificial intelligence} OR {software reusability} OR {user centred design} OR {dp industry} OR {societies and institutions} OR {computer supported cooperative work} OR {formal verification} OR {security of data} OR {middleware} OR {robots} OR {business data processing} OR {knowledge engineering} OR {decision making} OR {robot vision}) WN CV)

Figure 4. Search string. Document type (DT) was restricted to conference articles (ca), journal articles (ja) and articles in press (ip). The KY field refers to the subject, title and abstract of the article. The PN field refers to the publisher's name. The CV field refers to the controlled terms (i.e. standard keywords) of the article. The LA field refers to the language of the full text.

## 2.4 Methodology

A mapping process was used to identify the teamwork and taskwork activities reported in the 19 selected papers. Figure 5 illustrates the workflow of the methodology. A set of articles and texts (i.e. resources) related to teamwork in software development was selected. This selection process is identified as P0 in Figure 5. Then the teamwork activities of each resource were identified and retrieved. This process is identified as P1 in Figure 5. The retrieved teamwork activities are considered as “primary materials” since they are used in the next steps to reach the research objective. Using the taxonomy as reference, we mapped each identified activity to at least one relevant episode. This mapping is identified as the P2 process in Figure 5.

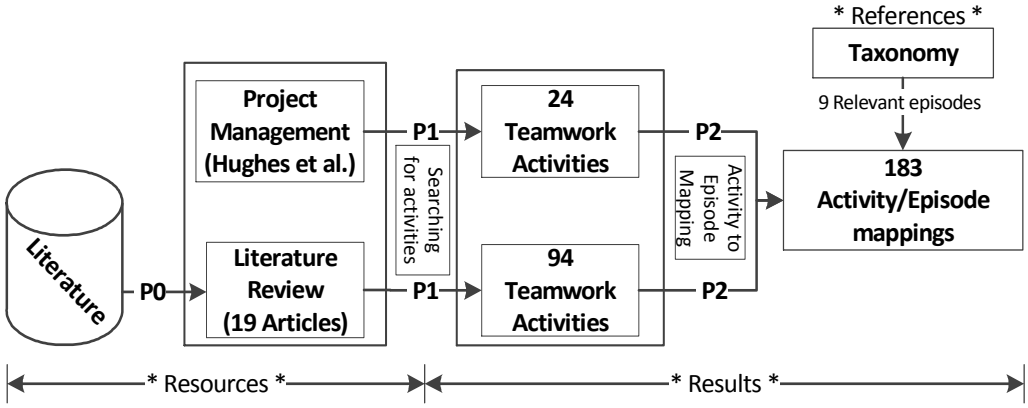


Figure 5. The processes (P0, P1, P2) illustrating teamwork activities discovery and episode mapping.

## 2.5 Mapping process

At this point of the workflow, all the teamwork activities have been identified and extracted from the resources (Process P1 in Figure 5). Using the reference model, each identified activity was associated to one or more relevant episode(s) (Process P2 in Figure 5). The identified teamwork activities have been extracted based on two view points: management and empirical studies perspectives.

### 2.5.1 Management perspective (Hughes and Cotterell Resource)

The mapping process started with the review of the teamwork activities described in Hughes et al. [1], in the form of the sub-steps of their framework. As mentioned earlier, Hughes et al. have highlighted all the steps where staffing concerns are important in their "stepwise framework" for software project planning. The objective here is to map all these teamwork activities and sub-activities from a project management perspective to at least one of the relevant episodes. In order to perform this process, all steps and sub-steps discussed in Hughes et al. [1] were identified and considered as teamwork activities (Process P1 in Figure 5). The activities were then mapped to the appropriate relevant episodes (Process P2 in Figure 5).

### 2.5.2 Empirical study perspective (Literature Review Resource)

As described earlier, the 19 selected articles were chosen from the literature databases. Each selected article was studied and then all reported teamwork activities were extracted (Process P1 in Figure 5). In this step, all identified activities are compared with the relevant

episodes derived from the taxonomy and are mapped to at least one of the episodes (Process P2 in Figure 5). If no mapping is possible, then that episode is considered as a "new" episode since it is not described in the SETTaxo [9].

We found three cases for the mapping of identified activities into relevant episodes.

- **"1:1" Correspondences:** This is the case where one extracted teamwork activity from the paper maps to a single specific relevant episode. For example, the identified activity "*Tracker - Manages the group diary, measures the group progress with respect to the estimations and tests score, manages and updates the boards*" [23], is directly mapped to the RM episode.
- **"1:new" Correspondences:** This is the case where the extracted activity cannot be matched into any of the relevant episode. A "new" episode is then defined to match the extracted activity.
- **"1:m" Correspondences:** This is the case where an extracted teamwork activity maps into many relevant/or new episodes. For example, a stand-up meeting in the Scrum process can be mapped to RM, BE, and FV episodes.

During the review, the following items were extracted for each article:

i. **Article Context:** Each article was assigned to one of these three categories: "Industrial", "Academic".

ii. **Article Objective:** The goals and objectives of the reported work from each article were extracted.

iii. **4P:** The proximity of each teamwork activity to one of the four aspects of "Project", "Product", "Process" or "People" were identified.

## 2.6 Results

### 2.6.1 Management Perspective (Hughes and Cotterell Resource)

After reviewing all the software project management "sub-steps" discussed in Hughes et al. [1], 24 teamwork activities were identified. Their definitions and detailed descriptions made possible a mapping of all the teamwork activities to at least one relevant episode. Table 2 shows the results of the mapping process. This table shows the retrieved activities and their corresponding mapped relevant episodes. The data in the 3rd column indicates the relevant episode name(s). For example "PP-RM" indicates that the related activity is similar to the PP and RM episode.

Table 2. Identified teamwork activities and mapped episode(s) for the Hughes et al. resource.[1]

| Teamwork activity | | Episode |
|---|---|---|
| Identify project scope and objectives | Identify objectives and practical measures of the effectiveness in meeting those objectives | FV |
| | Establish a project authority | FV-PP |
| | Stakeholder analysis - Identify all stakeholders in the project and their interests | FV |
| | Modify objectives in the light of stakeholder analysis | PP |
| | Establish methods of communication with all parties | PP |
| Identify project infrastructure | Identify relationship between the project and strategic planning | RM-PP |
| | Identify installation standards and procedures | PP |
| | Identify project team organization | RM |
| Identify the product and activities | Identify and describe project products (or deliverables) | FV-PP |
| | Document generic product flows | FV-PP |
| | Recognize product instances | PP-GA |
| | Produce ideal activity network | RM |
| | Modify the ideal to take into account need for stages and checkpoints | RM |
| Identify activity risks | Identify and quantify activity-based risks | PP-RM |
| | Plan risk reduction and contingency measures where appropriate | RM |
| | Adjust overall plans and estimates to take account of risks | RM-BE |
| Allocate resources | Identify and allocate resources | PP-RM |
| | Revise plans and estimate to take into account resource constraints | PP-RM-BE |
| Becoming a Team - Group feeling Development | Forming: The members get to know each other and try to setup some ground rules about behaviour. | ES-FV |
| | Storming: Conflicts arise as various members of the group try to exert leadership and the group's methods of operation are being established. | ES |
| | Norming: Conflicts are largely settled and a feeling of group identity emerges. | ES |
| | Performing: The emphasis is now the tasks at hand. | GA-IA |
| | Organizational structure definition | PP-RM |
| | Control ("Control" section as a part of "Monitoring and Control" subject) | RM |

### 2.6.2 Empirical studies perspective (Literature Review Resource)

After reviewing the 19 articles retrieved from the literature review process, a data table contained the following information:

i. All selected articles,
ii. Related context to each article (industrial, academic or theoretical),
iii. Objective of each article,

iv. Discovered teamwork activities after reading each article,

v. Mapped episodes to each activity,

vi. Scope of each activity according to the 4P perspectives (Project, Process, Product, People).

Table 3 shows an example of the results for one of the reviewed articles. The table uses the abbreviation "J" for proJect, "D" for proDuct, "C" for proCess and "P" for peoPle.

Table 3. Examples of identified teamwork activities and mapped episode(s) for one paper selected in the literature review.

| Paper Title | Instance found | Relevant Episodes | 4P |
|---|---|---|---|
| Application of tightly coupled engineering team for development of test automation software - a real world experience[24] | Each task in the project will be assigned to a pair of engineers. | PP3 | J |
| | Pairing is done based on the skill-set required for each task. | PP3-RM3 | P |
| | Pairing is done to create new knowledge on the team. Engineers new to a feature are paired up with engineers more knowledgeable about that feature. | TA3 | P |
| | Each pair shall be jointly responsible for the task in all phases – analysis, design, … | GA3 | C |
| | They will have the liberty to choose their own style of coupling within the parameters of joint responsibility so long as they synchronize their knowledge by the end of each day. | PP3-TA3 | P |
| | The structure of this network is dynamic in that the pairs are changed as new tasks arrive. | PP3-RM3 | JC |

The analysis of the selected articles resulted in 86 activities mapped into 141 episodes. Many activities were mapped into more than one relevant episode.

Table 4 shows some examples of activity/episodes performed during the mapping process.

Table 4. Examples of identified teamwork activities mapped to episodes.

| Instance of identified teamwork activities | Related Episode |
|---|---|
| "Customer: I had to follow and see that during implementation time people were working according to my stories." [23] | BE |
| "Scope/ Requirements capture & fulfillment" [25] | FV |
| "We use planning sessions to gather the entire team together and plan the next project release." [26] | PP |
| "Project management - Planning, monitoring and control" [25] | RM |
| "Reflection Meetings help us learn together and to apply our learning to improving our practices." [27] | TA |
| "On-site customer teams were visibly more effective" [28] | IG |
| "Team Morale-Motivation-Synergy, and Alignment & Communication" [25] | ES |
| "individual preparation" [29] | IA |
| "In charge of code effectiveness and correctness - Guides other teammates in the benefits of pair programming, enforces code inspection in pairs, searches for places in the code whose effectiveness requires improvement" [23] | GA |

## 2.7 Out of scope instances

Some teamwork activities found in the literature could not be mapped to an episode of the SETTaxo because they are outside of the scope of this research. In the following we present some examples which are out of the scope of this study because they are either not related to team objectives or done individually as a personal task.

- **Adjourning:** The group disbands. It happens when the activities of the team finish and team members disjoint and outspread [1]. This activity is not in the scope of this study since it is unrelated to the completion of the team objectives.
- **Group Decision Making.** This topic has been discussed in Hughes et al. [1] as a separate subject. In the context of the current work, it is observed that "Group decision making" can be divided into several relevant episodes, since it is not an atomic teamwork activity. The composition of these activities depends on the decision making goal. Do teammates discuss about the evaluation of past activities? In this case, it is a BE episode. Do they discuss about future activities to decide the feasibility of a certain task? In this case, it is a FV episode. Do they discuss about resources, project time line or costs and try to make a decision for controlling the project? In this case, it is a RM episode. As it can be seen, "Group decision making" activities are generic episodes which are typically missing too many context details to be assigned to one relevant episode.

- **Leadership.** During a software process or project, several activities, roles and artifacts involve "Leadership". Leaders and leadership refer to an informal role in the process. Several team activities can be assigned to a team leader based on the decision made by the team or by project managers. Thus the role by itself does not show a teamwork activity and cannot be considered in the current research work.
- **HSE** (Health, Safety and environment). In Hughes et al. [1], it is mentioned that team strategy on HSE should be clearly defined and the manager should be committed to this policy. However this matter is out of this work's scope since the technical objectives of a team is not completely dependent on HSE. These kinds of activities do not describe the interactions between team members.

## 2.8 Discussions

### 2.8.1 Management perspective (Hughes and Cotterell Resource)

The results show that all the teamwork activities retrieved from the management perspective were mapped to at least one episode without the need to define new episodes, as shown in Table 2. It shows that the taxonomy is able to cover and describe all identified teamwork activities from the project management point of view. As it can be seen, the RM and PP episodes have been used more frequently than the others. The reason is probably the proximity of the definitions of these two episodes to the project manager duties which is the focus of Hughes et al.'s book.

### 2.8.2 Empirical study perspective (Literature Review Resource)

Extracted activities from the literature review show that:

- **Demography:** About 80% of the 19 reviewed articles were in the Industrial context. Consequently, 70% of mapped episodes are in this context.
- **Episode:** As mentioned earlier, 94 extracted teamwork activities from literature review were mapped into 146 standard episodes which described in Table 5.

Table 5. Episodes appearance in retrieved activities (Resource: Literature)

| | BE | FV | PP | RM | TA | IG | ES | IA | GA | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| **Number of episodes** | 13 | 15 | 24 | 30 | 25 | 5 | 8 | 10 | 16 | **146** |

Table 5 shows that the PP, RM and TA episodes appear most often. When these results are compared with the results obtained from the Hughes et al.'s review, some similarity can be observed, which leads us to remark that the majority of reported empirical data in SE teamwork are about project management and the issues around it.

On the other hand the IG episode showed a low number of occurrences. It may be that the nature of the Information Gathering (IG) episode is such that it is not reported as often in teamwork processes. However, we found one study that reported a large number of IG episodes [19].

- **Combinational mapping:** Table 6 shows that the RM and PP episodes are mapped together for the same teamwork activity more often than any other pair of episodes. The PP and FV episodes are also combined often, with seven occurrences. On the contrary, the ES episode

Table 6. Episodes seen together with the same activity.

| | **BE** | **FV** | **PP** | **RM** | **TA** | **IG** | **ES** | **IA** | **GA** |
|---|---|---|---|---|---|---|---|---|---|
| **BE** | **3** | | | | | | | | |
| **FV** | 1 | **4** | | | | | | | |
| **PP** | 5 | 7 | **2** | | | | | | |
| **RM** | 4 | 4 | 10 | **10** | | | | | |
| **TA** | 1 | 1 | 4 | 5 | **11** | | | | |
| **IG** | 1 | 1 | 0 | 2 | 0 | **1** | | | |
| **ES** | 0 | 0 | 0 | 0 | 2 | 0 | **6** | | |
| **IA** | 1 | 0 | 3 | 0 | 1 | 0 | 0 | **2** | |
| **GA** | 2 | 0 | 3 | 3 | 4 | 0 | 0 | 5 | **3** |

rarely pairs with other episodes. The values on the diagonal in Table 6 show the number of occurrences that an activity was mapped to a single episode. For example, the ES episode is singly matched to its activity six times.

- **4P:** The proximity of each activity to one of the four perspectives of “proJect”, “proDuct”, “proCess” or “peoPle” were identified. Table 7 shows the results.

The values in this table show there are a few identified teamwork activities that are dealing with software products. The table also shows the following points:

i. The activities related to the "project" perspective are mostly mapped to the RM episode. The reason for this can be the proximity of their definitions. This observation is coherent with the result from the review of Hughes et al. [1] (see Table 2) where the RM episode had the highest value of mapping occurrences as well.

Table 7. Results from 4P point of view

| | **J** | **D** | **C** | **P** |
|---|---|---|---|---|
| **BE** | 7 | 3 | 9 | 3 |
| **FV** | 6 | 0 | 10 | 6 |
| **PP** | 9 | 1 | 12 | 7 |
| **RM** | 22 | 2 | 8 | 10 |
| **TA** | 5 | 2 | 12 | 16 |
| **IG** | 2 | 2 | 3 | 3 |
| **ES** | 1 | 0 | 1 | 8 |
| **IA** | 1 | 1 | 7 | 5 |
| **GA** | 4 | 0 | 13 | 5 |
| | **55** | **11** | **75** | **63** |

ii. Activity, role and artifact are three core concepts in the software management domain. The teamwork activities which are closer to the "process" perspective are mostly mapped to PP, TA and GA episodes. If we look at the definitions of these three episodes, PP is located in the transition phase and is closer to work products , the concept called "artifact" in the process management domain [30]. Two other episodes, TA and GA, are more practice oriented episodes. They are within the action phase and the work phase respectively, so they are closer to the "activity" concept of the process management domain.

The activities nearer to the "people" perspective are mostly mapped to TA and RM episodes. The rationale lies in the proximity of the definition of these episodes with the definitions of human interactions in the SE domain.

## 2.9 Threats to validity and reliability

Our methodology was validated using two methods in order to confirm that the employed analytical workflow is suitable for the research objectives. First, another researcher executed independently the two main processes (P1 and P2 in Figure 5), and second, a group of students

did the same. Two major risk items for this research project have been identified and are described below:

i. **Reporting manner:** In the step of teamwork activities extraction from articles (Process P1 in Figure 5), the extraction of activities depends on the way they are reported in the resources. The same teamwork interaction presented in two papers can be described differently. Therefore, retrieving the teamwork activities from the same article by two different researchers may cause different results. Validation with the independent evaluator and the student group showed that the P1 process provides consistent results. Therefore, this threat was not an issue for this study.

ii. **Mapping:** For the P2 process, the specific teamwork activity extracted from an article by the researchers can be mapped to different episodes. Validation showed that some problems occurred when the mapping was performed using an independent evaluator and a group of students. The activity/episode mapping should therefore be closely monitored by future researchers.

## 2.10 Conclusion

Reporting what a team is performing needs a technical vocabulary. The technical vocabulary used for these reports can be defined in a taxonomy of episodes. The SETTaxo is a software engineering taxonomy which presents a set of teamwork episodes. It is a tool which can be employed by researchers and software project managers for identifying and reporting all observed teamwork interactions. In this research, we have validated the SETTaxo through a mapping study.

Teamwork in software engineering has been viewed in this study from two perspectives: project management and empirical study. Project management is important since it is most frequently addressed in software teamwork literature. A literature review on empirical studies is also employed since scientific literature reports on many empirical studies on teamwork activities. Using these two view points, we identified and extracted a wide variety of teamwork activities.

The conclusions are itemized below:

- The SETTaxo showed that it can describe every kind of teamwork activities reported in the literature.
- All relevant episodes in SETTaxo were used with at least one teamwork activity.
- The RM and PP episodes were used more frequently than the others.

- The majority of reported empirical data in SE teamwork are about project management and the issues around it.
- The IG episode showed a low value of occurrence during the episode mapping process. The reason might be that the reporters are not aware of the importance of "information gathering" activities in team processes.
- The RM and PP episodes are mapped together for the same teamwork activity more often than any other pair of episodes, since they are very close to the project manager duties. A project manager typically manages the resources and revise the project plan at the same time, so (s)he performs RM and PP episodes conjointly. This can cause the reporters to consider these two episodes together.
- The PP and FV episodes are also combined often, since they are both in the transition phase and are respectively describing short and long term planning. This can cause the reporters to consider these two episodes together.
- On the contrary, the ES episode rarely pairs with other episodes, since during ES episodes the team members are not dealing with technical issues. It is typically observed alone in a team activity.
- There are a few extracted teamwork activities that are dealing with "software products".
- The activities which are often related to the "project" perspective are mostly mapped to the RM episode.
- The teamwork activities which are closer to the "process" perspective are mostly mapped to PP, TA and GA episodes, since these three episodes come from three different SETTaxo phases and can describe a minimal software process.
- The activities which are nearer to the "people" perspective are mostly mapped to TA and RM episodes. It shows that these two episodes are more people oriented than others.

In conclusion, the SETTaxo was able to describe all teamwork activities found in the selected literature. The SETTaxo also shows that the team process literature is mostly concerned with project management issues. The SETTaxo is therefore both complete and useful, as it can (1) describe every occurrence of team activities, and (2) provide useful information on the purposes of the team activities observed.

# CHAPTER 3:
# GENERAL DISCUSSIONS

As described earlier, the objectives of this research are:

- to deliberate the teamwork literature in the SE domain;
- to study the most recent understandings in the field; and
- to find a way to report an observed teamwork activity in order to highlight the strengths and weaknesses of a SE teamwork process.

## 3.1 Introductory sub-researches

In order to approach these research objectives, it is first needed to define a data gathering process. We need an efficient method to find and review the academic and experimental reports in the field. Classical systematic literature review methods enable the domain reviewers to identify, evaluate, select and synthesize high quality reports available in the literature in order to answer research questions. In my research context, a new version of the systematic literature review (iSR) method was used to identify the most relevant articles and to perform the final synthesis. The iSR method is an iterative approach of systematic literature reviews which performs the review through the use of multiple refinement iterations.

Systematic literature reviews (SLR) need a great deal of effort. The reviewers have limited time and resources to perform the full-scale review process. Many factors affect the amount of effort in a SLR:

- The structure of research questions,
- The background and motivation of reviewers,
- The structure of the domain under review,
- The allocated amount of time spent by reviewers,
- Etc.

As a result, our evaluations of the iSR process show that the Analysis activity takes the largest amount of time while the Synthesis activity requires the most mental effort.

As described earlier, we used the iSR as a tool to specify the most relevant articles and to perform the synthesis. Since pair programming is known as a teamwork practice in the SE

domain, a literature review process was employed in order to gather the definitions and main characteristics of pair programming as it is actually practiced in the field.

The review was limited to studies presenting empirical data from an industrial setting which could be related to real software development work. Selected studies and our own contributions to this issue have identified 38 context variables with a potential impact on Pair Programming effectiveness. We found that the original "two-person-at-one-computer" definition of Pair Programming covers a whole range of practices. Some of them might be very far from the type of Pair Programming that researchers originally had in mind. The literature shows that the purpose of having two individuals sitting together at one computer can vary widely.

Since our work on pair programming is a literature review, its threat to validity is related to repeatability of the selection process and data analysis. The main selection was performed by a single researcher. However, a second researcher validated random samples of the work performed by the first researcher. Initial samples showed low inter-rater agreement. As a result, selection was restarted until the samples presented a near-consensus on the selection process. The data analysis was performed by two researchers. Interpretations of the data extracted from the selected papers were discussed until a consensus could be reached.

We believe that by performing this pre-research study, the researchers will gain more experience on tools and methodologies which will be employed during the main research course.

## 3.2 Relation with the main research topic

As described in chapter 1, our studies show that Teamwork is an important topic which has been addressed in multiple references in the SE domain. Teamwork also has an outstanding role in software project management domain. A productive project manager also has been introduced as a person who is both a skillful manager and spirited leader [18].

In order to report accurately the team interactions, a vocabulary is needs to be defined. This vocabulary helps in the identification and diagnosis of team interactions. The work of Robillard and Lavallée [9] presents a useful and complete vocabulary for the SE domain. They viewed a software product from the team process perspective. Their work presents a new software engineering taxonomy of team processes with nine episodes which can be applied to report teamwork activities in the SE domain.

The taxonomy was validated through a mapping process between the defined episode and the literature on teamwork. Teamwork activities in the SE domain were observed from two

perspectives; a management perspective and an empirical study perspective. These two resources were selected because they are the only resources that can be found which address the team working in SE domain. The "software project management" book written by Hughes et al. was selected as the reference for management perspective and 19 related articles were selected from Compendex and Inspec databases for the empirical study point of view. It seems this number of articles can be enough for this research project since first of all the feasibility of the employed methodology is needed to be shown, while reviewing more article will not necessarily bring added value to current validation effort.

A mapping process is performed between teamwork activities found in the literature and teamwork episodes defined in the software engineering taxonomy. First, the articles selected were reviewed and the teamwork activities were identified. Then the identified activities were mapped to the relevant episodes of the taxonomy. As a result, 118 teamwork activities were extracted and were mapped to 183 teamwork episodes. The results show that all teamwork activities were mapped to at least one relevant episode. There was no need to define new episodes in the taxonomy; also, all relevant episodes were used with at least one teamwork activity.

In order to confirm that our methodology is suitable for the research objectives, first it was validated by another researcher who independently executed the two main processes (P1 and P2 in Figure 5). Secondly, a group of graduate and undergraduate students did the same.

Validation with an independent evaluator and a student group showed that the process of episode identification provides consistent results, while the activity/episode mapping needs to be closely monitored.

# CHAPTER 4:
# CONCLUSION

The validation process shows that employed methodology in teamwork study is reliable, albeit it needs some modifications and improvements in activity/episode assignment process. The submitted articles also show that the collected data, methodologies and results in teamwork study and in other two sub-studies (pair programming and iterative systematic review) are appropriate and reliable enough to be submitted in scientific journals.

A new research avenue can be considered as the "describability of pair programming styles using teamwork episodes". Is the SE taxonomy of team process appropriate enough to describe any style of pair programming? Answering to this question needs to investigate more and gather data and enough evidences.

As a feasibility study, we reviewed only 19 articles to retrieve teamwork activities in empirical study reports. It seems that for future works, this number of articles can be increased if the researchers face with one or more challengeable teamwork practices in SE domain.

In the following sections, the conclusion of each article is described separately.

## 4.1 Teamwork in software engineering

Describing the observations of team interactions needs a language. With the language's vocabulary, the observed components can be expressed via technical comprehensible terms. The "episodes" presented in Robillard et al. [9] and examined in this work are in fact the smallest significant part in a SE teamwork process. They can be compared to the words of a language.

Episodes (words) which are the components of the taxonomy (language) must have the ability to be employed by a user in order to describe any teamwork activities observed in the teams. Our research shows that all reported teamwork activities in the literature are describable via the taxonomy [9] and that no new episodes are required. The results also show that all nine standard episodes of the taxonomy have been used to describe at least one of the teamwork activities found in the literature.

The review from the 4P perspective of Project, Product, Process and People shows that the taxonomy is able to cover all teamwork activities from all perspectives. But the 4P model offers only a very high-level point of view. Hence the need for a taxonomy of team process episodes. This evaluation showed that some teamwork episodes are often evaluated in the selected

literature, and that some others are neglected. Using the taxonomy to evaluate the state of the literature provide a more accurate picture than the 4P model. Since the taxonomy presents a descriptive tool, it facilitates to report the teamwork observations.

The taxonomy is therefore both complete and useful. It is complete because it is able to succinctly describe all possible teamwork episodes. Furthermore, it is useful because it can qualify the state of the literature. It can therefore be concluded that the taxonomy is appropriate for the identification, description and report of teamwork activities in SE domain.

## 4.2 Pair programming

Since most of the time, the laboratory experiments do not demonstrate the complex interactions seen in the industry, the importance of field observations are outlined. Through 38 context variables which have the impacts on PP effectiveness, we consider only 21 of them as controllable and measurable, which is still too much for monitoring a PP study. One of the reasons is that PP was first introduced as a management practice in a light weight process. Nowadays, this practice has become a development practice. Therefore, a process interested by resource and schedule, has been shifted to a process interested by work responsibility, methodologies and artifacts quality assessments. This change of focus might explain the multidimensionality of PP studies. The question still exist that: “when two people work together, is it always PP?”. The literature shows that answer to this question covers a whole range of practices. Our research shows that the PP practitioners should first determine the context variables and purpose of pairing in order to benefit from pair programming.

## 4.3 Systematic literature review

Although domain experts have a key role in a systematic literature review process quality, our research shows that the iterative approach helps the SLR when working with domain novices. It is shown that the iSR generates better results and the reviewers can adjust their efforts and outputs gradually iteration by iteration. It has demonstrated that during an iterative literature review:

- The novice reviewers understand more about domain,
- Accordingly the structure of review process is evolved,
- The research questions may be rewritten,
- The selection process may need improvements,
- The extraction forms and analysis tables may be reviewed,
- The synthesis conclusion may be redesigned.

# REFERENCES


[1] B. Hughes and M. Cotterell, *Software Project Management*, 4th ed. McGraw Hill Education, 2005, p. 384.

[2] E. Salas, N. Cooke, and M. Rosen, "On teams, teamwork, and team performance: Discoveries and developments," ... *The Journal of the Human Factors ...*, vol. 50, no. 3, pp. 540–547, Jun. 2008.

[3] J. L. Dyer, "Team research and team training: A state-of-the-art review," *Human factors review*, pp. 285–3, 1984.

[4] M. Marks, J. Mathieu, and S. Zaccaro, "A temporally based framework and taxonomy of team processes," *Academy of Management Review*, vol. 26, no. 3, p. 356, Jul. 2001.

[5] E. Salas, D. E. Sims, and C. S. Burke, "Is there a 'Big Five' in Teamwork?," *Small Group Research*, vol. 36, no. 5, pp. 555–599, Oct. 2005.

[6] Y. Xiao, W. A. Hunter, C. F. Mackenzie, N. J. Jefferies, and R. L. Horst, "Task Complexity in Emergency Medical Care and Its Implications for Team Coordination," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 38, no. 4, pp. 636–645, Dec. 1996.

[7] E. E. Entin and D. Serfaty, "Adaptive Team Coordination," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 41, no. 2, pp. 312–325, Jun. 1999.

[8] J. E. Mathieu and S. B. Button, "An Examination of the Relative Impact of Normative Information and Self-Efficacy on Personal Goals and Performance Over Time '," pp. 1758–1775, 1992.

[9] P. Robillard and M. Lavallée, "Software team processes: A taxonomy," in *Software and System Process ( ...*, 2012, no. Icssp, pp. 101–109.

[10] J. Biolchini, P. G. Mian, A. Candida, and C. Natali, "Systematic Review in Software Engineering," no. May, 2005.

[11] D. Surian, N. Liu, D. Lo, H. Tong, E.-P. Lim, and C. Faloutsos, "Recommending People in Developers' Collaboration Network," *2011 18th Working Conference on Reverse Engineering*, pp. 379–388, Oct. 2011.

[12] F. W. Taylor, *Scientific Management (Google eBook)*. Routledge, 2003, p. 692.

[13] B. Hughes and Cotterell. Mike, *Software Project Management*, 4th ed. McGraw Hill Education, 2005, p. 384.

[14] J. R. Katzenbach and D. K. Smith, *The Wisdom of Teams: Creating the High-Performance Organization*. Harvard Business Press, 1992, p. 304.

[15] B. W. Tuckman and M. A. C. Jensen, "Stages of Small-Group Development Revisited," *Group & Organization Management*, vol. 2, no. 4, pp. 419–427, Dec. 1977.

[16] G. M. Parker, "Team Players and Teamwork: New Strategies for Developing Successful Collaboration," *John Wiley & Sons*. [Online]. Available: http://www.amazon.ca/Team-Players-Teamwork-Strategies-Collaboration/dp/0787998117/ref=sr_1_1?s=books&ie=UTF8&qid=1365509037&sr=1-1&keywords=team+players+and+teamwork+-jossy-bass.

[17] H. Eisner, "System Engineering: Building Successful Systems," *Morgan Claypool Publishers*, 2011. [Online]. Available: http://www.morganclaypool.com/doi/pdf/10.2200/S00349ED1V01Y201104ENG014.

[18] R. E. Fairley, *Managing and Leading Software Projects*. Los Alamitos, CA : Hoboken, N.J.: Wiley-IEEE Press, 2009, pp. 1–38.

[19] Y. Ton-that, P. N. Robillard, and M. Lavallée, "A Data Collection Technique for Observing Team Processes," in *under press*, 2013.

[20] D. L. Sackett, *Evidence-based medicine: how to practice and teach EBM, Volume 2*. Churchill Livingstone, 2000, p. 261.

[21] B. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering," in *26th International Conference on Software Engineering (ICSE'04)*, 2004, vol. 26, pp. 273–281.

[22] A. Zander, *Group dynamics: research and theory*, 2 nd. Evanston, IL: Row, Peterson and Company, 1962.

[23] Y. Dubinsky and O. Hazzan, "Roles in agile software development teams," ... *Programming and Agile Processes in Software ...*, 2004.

[24] A. Pandey, C. Miklos, and M. Paul, "Application of tightly coupled engineering team for development of test automation software-a real world experience," in *Computer Software ...*, 2003.

[25] A. Chatterjee and D. Israel, "Qualitative Survey-Based Content Analysis and Validation of Measures of Software Development Team Performance," *Information Intelligence, Systems, Technology ...*, pp. 299–310, 2011.

[26] P. Karsten and F. Cannizzo, "The creation of a distributed agile team," *Agile Processes in Software Engineering and ...*, pp. 235–239, 2007.

[27] M. Lamoreux, “Improving agile team learning by improving team reflections,” in *Agile Conference, 2005. Proceedings*, 2005.

[28] A. Wojciechowski, “Experimental Evaluation of’On-Site Customer'XP Practice on Quality of Software and Team Effectiveness,” in *On the Move to Meaningful …*, 2010, pp. 269–278.

[29] T. Thelin, “Team-based fault content estimation in the software inspection process,” in *... of the 26th International Conference on Software …*, 2004.

[30] P. Robillard, P. Kruchten, and P. D’Astous, *Software Engineering Using the Upedu*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[31] T. Dyba, E. Arisholm, D. I. K. Sjoberg, J. E. Hannay, and F. Shull, “Are Two Heads Better than One? On the Effectiveness of Pair Programming,” *IEEE Software*, vol. 24, no. 6, pp. 12–15, 2007.

[32] V. Balijepally, R. Mahapatra, S. Nerur, and K. H. Price, “Are two heads better than one for software development? The productivity paradox of pair programming,” *MIS Q. (USA)*, vol. 33, no. 1, pp. 91–118.

[33] J. E. Hannay, T. Dyba, E. Arisholm, and D. I. K. Sjoberg, “The effectiveness of pair programming: a meta-analysis,” *Inf. Softw. Technol. (Netherlands)*, vol. 51, no. 7, pp. 1110 – 22.

[34] K. Beck, *Extreme Programming Explained - Embrace Change*, Addison We. Reading, Massachusetts, 2000, p. 190.

[35] L. Williams and R. Kessler, *Pair Programming Illuminated*. Boston: Addison-Wesley Pearson Education, 2003, p. 265.

[36] S. Wray, “How pair programming really works,” *IEEE Softw. (USA)*, vol. 27, no. 1, pp. 50 – 5.

[37] W. C. Borman and S. J. Motowidlo, “Task Performance and Contextual Performance : The Meaning for Personnel Selection Research,” *Human Performance*, vol. 10, no. 2, pp. 99–109, 1997.

[38] H. Erdogamus, L. Rising, A. Main, A. Raybould, J. Stern, R. Morris, A. Glew, N. Coetzee, W. Miller, and S. Wray, “Responses to ‘How Pair Programming Really Works’,” *Software, IEEE*, vol. 27, no. 2, pp. 8–9, 2010.

[39] T. DeMarco and T. Lister, *Peopleware*. Dorset House Publishers, 1987.

[40] M. Unterkalmsteiner, T. Gorschek, A. K. M. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, “Evaluation and measurement of software process improvement-A systematic

literature review," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 398–424, 2012.

[41] M. Petticrew and H. Roberts, *Systematic Reviews in the Social Sciences*. Malden, MA: Blackwell Publishing Ltd, 2008, p. 336.

[42] N. Black, "Why we need observational studies to evaluate the effectiveness of health care," *British Medical Journal (BMJ)*, vol. 312, no. 7040, pp. 1215–1218, 1996.

[43] H. Gallis, E. Arisholm, and T. Dyba, "An initial framework for research on pair programming," in *Proceedings 2003 International Symposium on Empirical Software Engineering. ISESE 2003*, pp. 132 – 42.

[44] M. Ally, F. Darroch, and M. Toleman, "A framework for understanding the factors influencing pair programming success," in *Extreme Programming and Agile Processes in Software Engineering. 6th International Conference, XP 2005. Proceedings (Lecture Notes in Computer Science Vol. 3556)*, pp. 82–91.

[45] T. Dyba, B. A. Kitchenham, and M. Jorgensen, "Evidence-based software engineering for practitioners," *IEEE Software*, vol. 22, no. 1, pp. 58–65, 2005.

[46] E. Damiani and G. Gianini, "A non-invasive method for the conformance assessment of pair programming practices based on hierarchical hidden Markov models," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2007, vol. 4536 LNCS, pp. 123–136.

[47] M. A. Marks, J. E. Mathieu, and S. J. Zaccaro, "A temporally based framework and taxonomy of team processes," *Academy of Management Review*, vol. 26, no. 3, pp. 356–376, 2001.

[48] B. Kitchenham, S. Charters, D. Budgen, M. Brereton, M. Turner, S. Linkman, M. Jorgensen, E. Mendes, and G. Visaggio, "Guidelines for performing Systematic Literature Reviews in Software Engineering, Version 2.3," 2007.

[49] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, "Using Mapping Studies in Software Engineering," in *Proceedings of Psychology of Programming Interest Group*, 2008, vol. 2, pp. 195–204.

[50] B. Kitchenham, P. Brereton, and D. Budgen, "The educational value of mapping studies of software engineering literature," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, 2010, vol. 1, p. 589.

[51] L. Williams, "The Collaborative Software Process," University of Utah, 2000.

[52] J. T. Nosek, "The case for collaborative programming," *Commun. ACM (USA)*, vol. 41, no. 3, pp. 105 – 8.

[53] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, “Strengthening the case for pair programming,” *IEEE Softw. (USA)*, vol. 17, no. 4, pp. 19–25.

[54] D. Thomas and B. M. Barry, “Model driven development - The case for domain oriented programming,” in *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, 2003, pp. 2–7.

[55] M. Goldman and R. C. Miller, “Test-driven roles for pair programming,” in *Proceedings - International Conference on Software Engineering*, 2010, pp. 13–20.

[56] B. W. Tuckman, “Developmental sequence in small groups.,” *Psychological Bulletin*, vol. 63, no. 6, pp. 384–399, 1965.

[57] F. Maurer, “Supporting distributed extreme programming,” in *Extreme Programming and Agile Methods - XP/Agile Universe 2002 Second XP Universe and First Agile Universe Conference. Proceedings (Lecture Notes in Computer Science Vol.2418)*, pp. 13–22.

[58] R. Bednarik, A. Shipilov, and S. Pietinen, “Bidirectional gaze in remote computer mediated collaboration: Setup and initial results from pair-programming,” in *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, 2011, pp. 597–600.

[59] N. Salleh, E. Mendes, J. Grundy, and G. S. J. Burch, “An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model,” in *2010 32nd International Conference on Software Engineering (ICSE)*, vol. vol.1, pp. 577 – 86.

[60] J. E. Hannay, E. Arisholm, H. Engvik, and D. I. K. Sjoberg, “Effects of personality on pair programming,” *IEEE Trans. Softw. Eng. (USA)*, vol. 36, no. 1, pp. 61–80.

[61] K. S. Choi, F. P. Deek, and I. Im, “Exploring the underlying aspects of pair programming: the impact of personality,” *Inf. Softw. Technol. (Netherlands)*, vol. 50, no. 11, pp. 1114 – 26.

[62] T. Walle and J. E. Hannay, “Personality and the nature of collaboration in pair programming,” in *2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 203 – 13.

[63] S. S. J. O. Cruz, F. Q. B. Da Silva, C. V. F. Monteiro, C. F. Santos, and M. T. Dos Santos, “Personality in software engineering: Preliminary findings from a systematic literature review,” in *IET Seminar Digest*, 2011, vol. 2011, no. 1, pp. 1–10.

[64] M. Purvis, B. T. R. Savarimuthu, M. George, and S. Cranefield, “Experiences with pair and tri programming in a second level course,” in *Knowledge-Based Intelligent Information and Engineering Systems. 9th International Conference. KES 2005. Proceedings, Part II (Lecture Notes in Artificial Intelligence Vol. 3682)*, pp. 701 – 7.

[65] P. Dewan, P. Agarwal, G. Shroff, and R. Hegde, “Distributed side-by-side programming,” in *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering (CHASE 2009)*, pp. 48–55.

[66] P. Dewan, P. Agrawal, G. Shroff, and R. Hegde, “Experiments in distributed side-by-side software development,” in *2009 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2009)*, p. 10 pp. –.

[67] J. R. Nawrocki, M. Jasinski, L. Olek, and B. Lange, “Pair programming vs. side-by-side programming,” in *Software Process Improvement. 12th European Conference, EuroSPI 2005. Proceedings (Lecture Notes in Computer Science Vol.3792)*, pp. 28–38.

[68] K.-W. Han, E. Lee, and Y. Lee, “The Impact of a Peer-Learning Agent Based on Pair Programming in a Programming Course,” *IEEE Trans. Educ. (USA)*, vol. 53, no. 2, pp. 318 – 27.

[69] D. T. Sato, H. Corbucci, and M. V Bravo, “Coding Dojo: an environment for learning and sharing agile practices,” in *Agile 2008*, pp. 459 – 64.

[70] A. Gaspar and S. Langevin, “Restoring ‘coding with intention’ in introductory programming courses,” in *SIGITE’07 - Proceedings of the 2007 ACM Information Technology Education Conference*, 2007, pp. 91–98.

[71] B. Kitchenham, S. Charters, D. Budgen, M. Brereton, M. Turner, S. Linkman, M. Jorgensen, E. Mendes, and G. Visaggio, “Guidelines for performing Systematic Literature Reviews in Software Engineering, Version 2.3,” 2007.

[72] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, “Using Mapping Studies in Software Engineering,” vol. 2, 2007.

[73] F. Garcia, G. Canfora, A. Cimitile, M. Piattini, and C. A. Visaggio, “Evaluating performances of pair designing in industry,” *J. Syst. Softw. (USA)*, vol. 80, no. 8, pp. 1317 – 27.

[74] A. L. Strauss, “Qualitative Analysis for Social Scientists,” *Cambridge University Press*, 1987. [Online]. Available: http://www.cambridge.org/gb/knowledge/isbn/item1134739/?site_locale=en_GB. [Accessed: 24-Mar-2013].

[75] A. Rainer and S. Beecham, “Supplementary guidelines and assessment scheme for the use of evidence based software engineering,” *Hatfield, UK, University Of Hertfordshire Technical …*, 2008.

[76] P. Brereton, “A Study of Computing Undergraduates Undertaking a Systematic Literature Review,” *IEEE Transactions on Education*, vol. 54, no. 4, pp. 558–563, Nov. 2011.

[77] A. Rainer and S. Beecham, “A follow-up empirical evaluation of evidence based software engineering by undergraduate students,” in *Electronic Workshops in Computing*, 2008.

[78] M. Jørgensen, T. Dybå, and B. Kitchenham, “Teaching Evidence-Based Software Engineering to University Students.”

[79] A. Rainer, S. Beecham, and C. Sanderson, “An assessment of published evaluations of requirements management tools,” pp. 1–10, 2007.

[80] A. Rainer, T. Hall, and N. Baddoo, “A preliminary empirical investigation of the use of evidence based software engineering by under-graduate students,” 2005.

[81] M. T. Baldassarre, N. Boffoli, D. Caivano, and G. Visaggio, “A Hands-On Approach for Teaching Systematic Review,” pp. 415–426, 2008.

[82] B. Kitchenham, “Procedures for Performing Systematic Reviews,” 2004.

[83] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic Mapping Studies in Software Engineering,” pp. 1–10, 2007.

[84] P. Brereton, M. Turner, and R. Kaur, “Pair programming as a teaching tool: a student review of empirical studies,” in *2009 22nd Conference on Software Engineering Education and Training. CSEET 2009*, pp. 240 – 7.

[85] B. S. Bloom, *Taxonomy of Educational Objectives: The Classification of Educational Goals.* Longman Group United Kingdom, 1969.

[86] B. J. Oates and G. Capper, “Using systematic reviews and evidence-based software engineering with masters students,” pp. 1–9.

[87] M. Lavallée and P. N. Robillard, “The Impacts of Software Process Improvement on Developers : A Systematic Review,” pp. 113–122, 2012.

[88] T. Dybå and T. Dingsøyr, “Empirical studies of agile software development: A systematic review,” *Information and Software Technology*, vol. 50, no. 9–10, pp. 833–859, Aug. 2008.

[89] D. S. Janzen and J. Ryoo, “Seeds of Evidence: Integrating Evidence-Based Software Engineering,” *2008 21st Conference on Software Engineering Education and Training*, pp. 223–230, Apr. 2008.

[90] K. R. Felizardo, M. Riaz, M. Sulayman, E. Mendes, S. G. MacDonell, and J. C. Maldonado, “Analysing the Use of Graphs to Represent the Results of Systematic Reviews in Software Engineering,” *2011 25th Brazilian Symposium on Software Engineering*, pp. 174–183, Sep. 2011.

[91] D. S. Cruzes and T. Dybå, “Research synthesis in software engineering: A tertiary study,” *Information and Software Technology*, vol. 53, no. 5, pp. 440–455, May 2011.

[92] G. a Miller, “The magical number seven, plus or minus two: some limits on our capacity for processing information. 1956.,” *Psychological review*, vol. 101, no. 2, pp. 343–52, Apr. 1994.

[93] F. O. Bjørnson and T. Dingsøyr, “Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used,” *Information and Software Technology*, vol. 50, no. 11, pp. 1055–1068, Oct. 2008.

[94] Human Performance Research Group, “NASA Task Load Index (TLX) v1.0 Manual,” California, 1986.

# ANNEXES

## ANNEX 1: THE ANATOMY OF THE PAIR: A MAPPING STUDY OF HOW PAIR PROGRAMMING IS DEFINED IN PRACTICE

*Authors: Mathieu Lavallée, Reza Mirsalari, Pierre-N. Robillard.*

*This article was submitted to TSE journal.*

*My contribution to this article:*

a. *Performing a mapping study to fine the state of the art of pair programming and all the styles and context variables reported in the literature.*
b. *Refining the results of applying a specific search string on article data bases in order to compile the most related articles.*
c. *Analysing and specifying the context and type of each article, e.g. industrial/academic contexts and experimental/observational type articles.*
d. *Continuously, revising the paper structure.*

My specific contribution to this paper is marginal.

**Abstract**— Pair Programming was initially defined simply as two programmers sitting at one computer. This definition has since evolved into variants, to the point that the practice is no longer unified. This literature review gathers the definitions and main characteristics of Pair Programming as it is actually practiced in the field. The review uses the Kitchenham literature review methodology common in software engineering research. The review shows that few studies can be mapped to the development projects undertaken in the industry. Selected studies show that many context variables are not appropriately reported, making inter-study comparison difficult. Previous work and our own contributions to this issue have identified 38 context variables with a potential impact on Pair Programming effectiveness. We found that the original "two-person-at-one-computer" definition of Pair Programming is very large and covers a whole range of practices. Some of these practices might be very far from the type of Pair Programming that researchers originally had in mind. The literature

shows that the purpose of having two individuals sit together at one computer can vary widely. This difference in pairing purposes could explain why the benefits of pairing are contradictory from one study to another.

**Index Terms**—Pair programming, programming teams, software process models, literature review.

# 1. INTRODUCTION

The simple question, "are two heads better than one?", posed by Tore Dyba et al. in 2007 [31] and by Balijepally et al. in 2009 [32], captures the fundamental issue underlying Pair Programming (PP) effectiveness. Are there any real benefits to having two programmers working on a single computer? Despite the extensive body of PP studies, there are as yet no useful answers to this question. As Hannay et al. wrote in their meta-analysis of PP effectiveness [33]:

*"The question of whether pair programming is better than solo programming is not precise enough to be meaningful, since the answer to that question in the present context is both 'yes' and 'no'. On the basis of the evidence from this review, the answer is that 'it depends'".*

Hannay et al.'s conclusion is that moderating factors, such as expertise and task complexity, are often not considered in PP studies, resulting in large variations in the results. The context of the research is too often neglected, and many studies have poor external validity.

Is the cause for this external validity problem limited to the methodology used in PP studies, or is there a more fundamental problem with the definition of PP itself? While many studies base their definition of PP on the work of either Beck [34] or Williams and Kessler (W&K) [35], some research on practitioners shows that *"no one [has] claimed that it [is] practiced as written in the books"* [S22][4]. The question is, therefore, what definitions of PP are being used by practitioners in the industry? Put another way, by Stuart Wray [36]:

*"Programmers label a wide variety of practices under the 'pair programming' umbrella. Thus, before our community can sensibly discuss how pair programming works, we first need to establish exactly what it is."*

## 1.1 A PAIR PROGRAMMING LEXICON

Before embarking on the multiple definitions of PP, we must first define the many terms and expressions used in the literature. The PP practice is characterized by a number of unique

[4] References with the prefix 'S' are those that have been selected by the literature review. These references can be found in the Appendix.

features and challenges which must be understood in order to be managed. The following lexicon describes these terms:

- **Roles**: The roles describe what the developer at the keyboard and mouse must do, and what the other developer must do. They are defined by the responsibilities and competencies required. W&K [35] have defined them as 'driver and navigator', although the validity of this metaphor is currently being questioned [S9].
- **Solo programming**: This is work performed alone, without a pair partner. Most studies show a mix of PP and Solo work. W&K insist that pairing should be encouraged, but not enforced ([35], p52).
- **Backup behavior (BB)**: Good developers show good 'organizational citizenship behavior' [37], in that they have a natural inclination to assist their colleagues when asked. BB implies two developers working at one computer, but the relationship between them is different from what it is in PP sessions. While PP is motivated by shared problem-solving, BB is motivated by the transfer of a known solution from the helper to the individual requesting it.
- **Pair switching**: This occurs when the developer at the keyboard and mouse gives control of the computer to his/her pair partner. It represents an exchange of roles within the pair ([35], p. 4).
- **Pair rotation**: This occurs when a developer leaves his/her current pair to pair with someone else. It ensures that a developer pairs with different developers during the course of the project [35], [36].
- **Pair pressure**: This mechanism ensures that pair partners are more focused on the work to be done and on the process to follow, because everything they do individually is observed by their pair partner [35].
- **Pair fatigue**: Pair rotation ensures that a fresh pair of eyes observes the problem at hand. This new look can help developers find a solution when they are stuck. However, its effect is limited, since, after a while, *"the things [developers] notice become more similar. Eventually, the benefit from two pairs of eyes becomes negligible."* [36]. This phenomenon is called 'pair fatigue'.
- **Partner burnout**: Developers in pairs are much more focused than developers working alone. To quote Andy Glew [38] : *"I find that eight hours of pair programming is more*

*exhausting than 16 hours of solo work. You simply work harder while pair programming."* PP session management must take this factor into account.

Table 8. Three definitions of pair programming according to the literature

| Characteristic | **Beck** [34] | **Williams and Kessler** [35] | **Wray** [36] |
|---|---|---|---|
| **PP definition** | Two programmers at one machine | Two programmers working at one computer | – |
| **Roles** | Implementer and strategist | Tactical driver and strategic navigator | A "jelled" team works at the same level of abstraction ("tag team") |
| **Pair switching** | Hinted | Important to switch roles periodically | – |
| **Pair rotation** | A couple of times a day | A few hours to a few weeks | Every two hours, to prevent pair fatigue |
| **Rotation pool** | – | A developer should have between 9 and 11 potential partners | – |
| **Tolerance of solo work** | Solo work forbidden | Pairing should be encouraged, but not enforced | As little as possible, to benefit from pair pressure |
| **Physical arrangement** | Two programmers with one keyboard, one mouse, and one monitor. Keyboard easy to shift from side to side | The same as Beck, but with a minimum 17-inch monitor. Solo workers capable of working without being disturbed by pair discussions | – |
| **Collocation** | Mandatory | Distributed pairing possible, but not recommended | - |
| **Intra-pair communication** | – | Maximum of 45 seconds of silence | As much as possible, to benefit from a mutual rubber plant effect |
| **Pair scheduling** | – | Core PP hours defined, to accommodate flex time | – |
| **Pair assessment** | – | Workers' assessment to include peer evaluations | Managers can take advantage of the implicit identification of expertise by the pair partners |
| **Partner selection** | – | Pair with the individual most likely to have the knowledge required | – |
| **Tasks** | XP tasks focus on coding and unit test writing | Pairing useful for any task, from specification, to design, to testing. | – |

- **Rubber plant effect**: Vocalizing a problem can help the speaker better understand it, because of the need to synthesize it first to perform the vocalization. Similarly, describing a problem out loud, even when the developer has only a rubber plant for company, can help him/her to better understand the problem. This effect is thought to be one of the benefits of PP, because the two developers are constantly vocalizing problems and possible solutions to one another [36].

- **Jelled PP team**: A pair is in a jelled state when the pair partners are familiar with each other's work approach and idiosyncrasies. According to few studies [S25, S26, S28], a pair must work together for at least 10 hours to be familiar enough with each others to be considered jelled.
- **Distributed PP**: There is an increasing number of studies dedicated to PP in a distributed setting. The key principles remain the same: One user has read/write access to the work performed, while another, a distant user, has read-only access. This setting requires the use of a communication tool, and introduces specific interaction challenges [S2]. This will be discussed in more details in section 5.4.

Table 8 presents three core definitions of PP for each of the characteristics. The first definition is based on the works of Beck [34] in the context of the Extreme Programming (XP) process. The second definition is a relaxed version of PP, as defined in the works of W&K [35]. The third definition, which is based on PP as seen from an industrial environment perspective and described in a short paper by Wray [36], presents the core mechanisms of pairing and what the pairing context must ensure. A dash means that this aspect of the definition is not discussed by the authors. These three definitions are discussed in more detail in the following subsections.

**1.2 Pair Programming and Extreme Programming**

What exactly is Pair Programming? The roots of the practice date back to the 1950s ([35], p. 8), but credit for formalizing Pair Programming (PP) is typically attributed to Kent Beck [34]. The details of the practice were first described within the context of Extreme Programming (XP). The key characteristics of pairing within XP are presented in the Beck column of Table 8.

Beck's definition of PP is very formal, as there is no room for solo programming in this context. All code must be written by pairs. The rationale behind such formality is justified by W&K [35]:

*"We believe pair programming is an integral part of XP, and it is dangerous to do XP without doing pair programming. One primary reason is that the pairs keep each other honest. XP is a minimalist approach, so it is essential that many of the practices actually get done."*

The key aspect of mandatory pairing is what the literature calls "pair pressure", the fact that developers are more careful about the quality of their work because their work is continuously

assessed by their pair partner. Wall-to-wall pairing in the theoretical XP context is mandatory by necessity. But what happens when we move out of this context?

### 1.3 Pair Programming at Large

W&K's book on PP [35] is dedicated to the application of the practice at large; including outside the context of the formal XP process. In a more documented process, pair pressure is no longer a necessity, and it is no longer mandatory that all code be written in pairs. PP is still recommended for more complex tasks ([35], p. 15), but solo programming is tolerated. This relaxed form of PP is defined by the characteristics presented in the W&K column of Table 8.

Therefore, while PP as defined by W&K remains an institutionalized practice, solo programming can occur when it is warranted. But, where is the line between institutionalized practice and casual use? According to W&K ([35], p. 33):

*"There are […] two levels of pair programming: One is a very casual, non-invasive use of the practice. [The other is a] prevalent use of pair programming throughout a team, where pairing is an integral part of the team dynamics and the team development practices [...]."*

### 1.4 Frontiers of Pair Programming

Software development is essentially a social activity: The software product is no longer the product of a single individual, but rather the product of a whole team. As human beings, software developers are social beings, and will assist each others as needed. Good developers do not need to be told to pair, in order for them to help each other: they will do so naturally [37]. As described earlier, this natural behavior of good developers is called Backup Behavior (BB).

The question here is, what are the differences between PP and BB? Moreover, are these differences significant, or can a team be called a PP team when it only pairs during BB interactions? We believe that the two types of interaction, PP and BB, are very different.

We theorize that, in jelled PP teams, the partners share their knowledge on an equivalent level, as peers. This exchange is motivated by the shared objective of the task. Formed pairs can last for a long time, from a few hours to a few weeks [35].

A less equivalent form of information exchange would be PP teams formed of a mentor and a novice [S1], [S20], [S22], [S28]. This type of pairing is often used to integrate new members into the development team. Although the partners work together on a problem, most of the information is transferred one way, from the expert to the novice. Mentoring pairs typically last

for a shorter time [S22], and the pairing ends once the new member has been successfully integrated into the team.

BB is also mostly a one way transfer of information, and it occurs when a developer requests the assistance of another. BB interactions can last from a few seconds ("Where is the configuration file?"), to a few minutes ("Can you show me how to import the project from the source control?"), and sometimes up to a few hours ("Show me how to set up a new production environment on the server.").

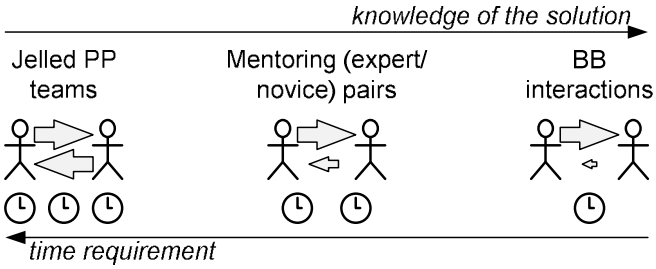


Figure 6. Difference in information exchange and exchange duration in jelled PP teams, mentoring pairs and BB interactions.

Figure 6 illustrates the major differences between jelled PP teams and BB interactions with mentoring sessions in terms of the amount of information exchanged, represented by the knowledge of the solution arrow (top), and the duration of the exchange, represented by the time requirement arrow (bottom). For example, in a jelled team, the exchange of information between the partners should be equivalent, as represented by the two large arrows pointing towards the partners on the left-hand side of Figure 6. BB interactions are illustrated by a small information arrow pointing towards the helper and a larger information exchange arrow pointing towards the individual being helped. These differences should be taken into account when pairing data are analyzed to assess the effectiveness of pair programming.

In practice, this could be very difficult. We theorize that these three types of pairs are present in every software development team, in a continuous spectrum. A team environment that does not support PP will contain a few sessions where two developers work together and share a problem solving objective (i.e. perform an implicit PP session). Similarly, a team environment in which 100% pairing is enforced will show some evidence of BB interaction within the pairs. This theory is in line with the seminal work of De Marco and Lister ([39], cited by [7]), which

states that the overall effort of a software development project is split as follows: 30% by one developer working alone, 50% by two developers working together, and 20% by more than two developers working together in a meeting. The 50% effort expended by two developers working together could be a mixture of PP, mentoring, and BB.

Why is this PP – BB spectrum important? It is important because true pairing, where two developers working together on the same problem, might not occur as often as we think. For example, if 10% of all development effort is expended in pairs, as is reported in some studies, then a significant proportion of this effort might actually be BB. Similarly, a team might report high pairing levels because it has suddenly expanded in size and needs to engage in frequent mentoring or BB sessions. Pairing levels show how often two developers work together, but not the nature of the interactions between the pair partners.

### 1.5 What is Pair Programming?

As we saw in the previous section, the frontier between PP and other collaborative behaviors is fuzzy for many researchers in the field. The definition of PP used by some researchers would not be recognized by others. While the exact position of this frontier is of interest to the taxonomist within us, it is irrelevant at this stage. What we need to do is to be able to accurately assess the true benefits of PP, which requires that we identify comparable studies. In order to achieve this, we need to determine the precise definition of PP used, which we do by answering the following two research questions:

- RQ1: What context details are needed to properly identify the various definitions of PP?
- RQ2: What are the definitions of PP used in studies performed in the field?

The objective of this paper is to answer these two questions. We start by presenting the current recommendations for PP studies in section 2. We also present our answer to the first research question, in the form of our own recommendations for future PP studies. We then present, in section 3, the literature review method used to map the current state of PP research. The results and the current state of PP research are presented in section 4. Finally, in section 5, a discussion is presented on the current state of PP research, which is aimed at answering the second research question.

## 2. CURRENT RESEARCH IN PAIR PROGRAMMING

The state of research in PP is similar to the state of research in software engineering in general. Both suffer from numerous flaws, among them, as reported by Unterkalmsteiner et al.,

confounding factors and incomplete context definitions [40]. Confounding factors are the research factors that influence the observations made, but are not considered. Incomplete context definitions are those that lack detail on the reasons for, and approaches to, PP use.

**2.1 On Confounding Factors**

Many research papers on PP are based on short experiments performed in a laboratory context. This is problematic, as learning to work in a pair is not a trivial task. Empirical evidence indicates that it takes at least 10 hours for a pair to attain a jelled state [S25, S26, S28]. Unfortunately, most of the current PP research involves observing developers unfamiliar with PP, and for a very short time, often measured in minutes.

According to Petticrew and Roberts, in their book on systematic literature reviews [41], while controlled laboratory experiments can have strong internal validity, the laboratory context might not be representative of what happens to the population in a natural context. They add that *"observational data [...] may be more typical of those who deliver and receive the intervention outside of research settings"* ([41], p. 185).

They also point out ([41], p. 185) that, based on previous work by Black [42], controlled experiments are not always the highest form of research study, since some domains are highly dependent on context variables. In concordance with this remark, there does seem to be agreement in the current PP research on the importance of context variables [43], [44].

This concern for industrial context research over laboratory experiments, which applies to the research community at large, has been confirmed by Hannay et al. [33] for PP research in particular. Some of the conclusions of their meta-analysis on the effects of PP no longer hold when certain critical context variables are considered. In their meta-analysis, about 55% of their constructs for quality, duration, and effort results were based on results obtained in an academic setting, and only 45% in industrial settings. As Dyba et al. argue in another paper [45]:

*"Medical evidence is based on rigorous studies or therapies given to real patients requiring medical treatment; laboratory experiments aren't considered to provide compelling evidence. This implies that SE shouldn't rely solely on laboratory experiments and should attempt to gather evidence from industrial projects, using observation studies, case studies, surveys, and field experiments. These empirical techniques don't have the scientific rigor of formal randomized experiments, but they do avoid the limited relevance of small-scale, artificial SE experiments."*

The lack of empirical evidence and the unclear definition of PP used in studies were also cited by Damiani and Gianini [46] in 2007 as a inadequate basis for making claims on the benefits of PP:

*"PP, in its different variants, has been claimed to yield, as a part of the extreme programming process, higher quality software products in less time. The claim is supported by anecdotal evidence and by empirical studies. However a more systematic study of the practice would be desirable: one based on real development settings, linking the degree of adherence to the practice to the quality level of software."*

The lack of a clear definition of PP in most empirical research has also been noted by Stuart Wray [36]:

*"Such misunderstanding shows that we can't take a claim that developers are pair programming at face value; they might not be doing what experienced and effective pair programmers actually do."*

The participation of students seems problematic for the study of PP effectiveness, as we know that the dynamics of a pair can vary wildly, depending on the level of expertise of its members [33], [35]. Research shows that the members of pairs made up of two novices, a novice and an expert, and two experts do not interact in the same way.

The use of the laboratory experiment format for the evaluation of PP effectiveness also seems problematic, especially when the task complexity is trivial compared to that of the corresponding professional task performed in the real world [33]. It is doubtful that a one-hour task involving coding and testing a polynomial function, for example, can be generalized to an actual professional task.

The call for more industrial observations in software engineering is nothing new. However, this call is much more important for PP research, because of the time required for the pairs to learn and to work with one another, in order to attain this jelled state.

**2.2 On Incomplete Context Definitions**

Since PP is no longer a unified concept, as demonstrated in the first section of this paper, the effectiveness of PP now depends on the ability to compare similar definitions of the practice. It also depends on comparing similar contexts of PP use. For such meta-analyses to be possible, additional information must be provided by PP researchers.

Stuart Wray advocates the use of a context checklist, on which the details of how PP is actually performed would be presented [36]. Gallis, Arisholm, and Dyba developed such a checklist for empirical PP studies in 2003 [43]. This research framework was augmented in 2005 by Ally, Darroch, and Toleman [44] to account for more context variables deemed important by the authors. In our opinion, this checklist is still incomplete, as it doesn`t account for the differences between PP definitions. Table 9 shows the source of each of the context variables.

The 'C' column indicates whether the variable is controllable in a standard field study. The 'M' column indicates whether the variable is measurable. In a controlled experiment, the observed phenomenon (e.g. PP effectiveness) is affected by the controlled variables, measurable noise variables and unknown effects. A variable both controllable and measurable can be used as a control variable in such a controlled experiment (e.g. pairing proportion). A non-controllable but measurable variable will be labeled as noise; its effect on the output can be evaluated, but it cannot be easily planned for (e.g. intra-pair communication). A non-controllable and non-measurable variable falls in the unknown effects: Variables which plausibly have effects on the output but whose effects cannot be directly evaluated (e.g. interpersonal relations). This identification of controllable and measurable variables was not included in the Gallis and Ally papers.

Readers are referred to the papers of Gallis [43] and Ally [44] for a more detailed explanation of their respective context variables. In the table, the sources labeled NEW are the ones proposed by the authors of this paper, and explained below:

- **Pairing ratio:** How much of the task is actually performed by two programmers working as a pair? A number of papers have claimed to study PP, but did not validate that the developers really worked in pairs [S2, S14].
- **Pair switching**: How often do the paired developers change places at the keyboard? Is pair switching mandatory or performed in ad hoc fashion?
- **Intra-pair communication**: How often do the pair partners communicate with one another? While an exact value might be difficult to establish, researchers should verify that both pair partners contribute actively to the work.
- **Size of the rotation pool**: How many potential pair partners the developers have in the study? A single potential pair partner implies that no pair rotation takes place, which means that a potential benefit of PP will not be observed.

Table 9. Context variables and their status (C: Controllable, M: Measurable) found in empirical PP studies

| Context Variables | C | M | Source |
|---|---|---|---|
| **Pairing ratio** | √ | √ | **NEW** |
| Education and experience | | √ | Gallis |
| Personality | | | Gallis |
| Interpersonal relations | | | Ally |
| Programmer resistance to PP | | | Ally |
| Roles | | √ | Gallis |
| Shared responsibility | | | Ally |
| **Pair switching** | √ | √ | **NEW** |
| Communication channels | √ | √ | Gallis |
| **Intra-pair communication** | | √ | **NEW** |
| Pair rotation | √ | √ | Gallis |
| Rotation accounted in project management | √ | √ | Ally |
| Effectiveness of the pair | | √ | Ally |
| Effective knowledge sharing | | | Ally |
| Threatening feeling for the pair partner | | | Ally |
| Type of development activity | √ | √ | Gallis |
| Size of the task | √ | √ | Gallis |
| Comlpexity of the task | √ | √ | Gallis |
| Duration of the task | √ | √ | Gallis |
| PP task fit | | | Ally |
| **Pairing objective** | √ | √ | Ally/**NEW** |
| Software development process | √ | √ | Gallis |
| Project shcedule | √ | √ | Ally |
| Software development tools | √ | √ | Gallis |
| Tools of distributed PP | √ | √ | Ally |
| Physical disposition of the workspace | √ | √ | Gallis |
| Solitude and privacy | √ | √ | Ally |
| Team building and pair management | | | Ally |
| **Pair scheduling** | √ | √ | **NEW** |
| **Partner selection** | √ | √ | **NEW** |
| Human resource management | | | Ally |
| **Team changes** | √ | √ | **NEW** |
| **Size of the rotation pool** | √ | √ | **NEW** |
| Pair accountability | | | Ally |
| **Pair assessment** | √ | √ | **NEW** |
| Customer resistance to PP | | | Ally |
| Organizational culture towards PP | | | Ally |
| Collective code ownership focus | | | Ally |

- **Pairing objective**: The main objective of the pair sessions should be reported. Is the purpose of the pair session to solve a problem (jelled PP), to introduce a new member to the team (mentoring), to assist a developer with a specific problem (BB)? Or was the pair formed for another specific purpose? Ally introduced this context variable, but it was limited to the identification of mentor pairing sessions
- **Pair scheduling**: Does management ensure that time is set aside for pairing in their work schedule? Are developers actually available for pairing during that time?
- **Partner selection**: How are pairs formed? Is pairing an opportunistic or a managed process?
- **Team changes**: New developers joining the team can have an impact on the nature of the pairing activities. A team with many new hires might do more mentors pairing than a stable team.
- **Pair assessment**: Are peer evaluations used in reviewing the performance of paired developers?

As shown in Table 9, there are numerous context variables with the potential to impact PP effectiveness. Again, in our opinion, it is doubtful that an academic experiment can accurately replicate the complexity of real PP work. While controlled experiments have their place in exploratory software engineering research, observational studies performed in the industry should be the focus of reviews on PP effectiveness.

Additionally, we found that some of the context variables identified in Table 9 are neither easily controllable nor measurable. Using these variables in

case studies makes it difficult to assess their impact on PP. The 38 variables listed in Table 9 are an inventory of the various parameters that have been considered in the various papers on PP. However the 21 context variables that are both controllable and measurable should be reported in any study on PP. The four context variables, which are not controllable but are still measurable, could provide some interesting insight on the individual team members. Finally, the 13 non-controllable and non-measurable variables listed in Table 9, originally introduced by Gallis and Ally, belongs to the domain of team process dynamics, more precisely to the concept of emergent states [47]. An emergent state is a mediating mechanism defined as the cognitive, motivational and affective states having a potential impact on the process under study. Some of these emergent states are measurable and can be controlled in empirical studies in the field of psychology but are not likely to be part of traditional methods in software engineering studies.

# 3. METHODOLOGY

Our approach to collecting PP studies is an iterative method based on the processes developed by Kitchenham et al. for systematic literature reviews (SLR) [48] and evidence-based software engineering (EBSE) [21], in order to demonstrate the results of a mapping study or a scoping review [49]. However, as Budgen et al. [49] and Kitchenham et al. [50] point out, there is no widely recognized process for mapping studies. The approach we use to map the current research literature on PP is described in the following section.

## 3.1 Selection Process

The search process was performed on the Compendex, Inspec, ACM, and IEEE Xplore databases. These databases were chosen because they include the most relevant journals and conferences in software engineering. The search string "pair programming" returned 1,362 titles.

Duplicates, proceedings titles, advertising tracts, research proposals, and keynote presentations were removed from the selection. We also limited the selection to papers published from 2000 onwards, the year of publication of Beck's book [34]. Titles unrelated to software engineering were also removed. Papers of five pages or less were removed, on the basis that they typically do not describe the context of the study sufficiently well. In addition, we removed those for which the full text was not available, as well as papers not available in English.

The selection process resulted in a total of 233 papers.

### 3.2. Refinement of the Selection

We found, however, that many of these 233 papers did not show a validation in a realistic setting. As demonstrated previously, a proper evaluation of PP requires field observations. We therefore refined our selection, and show the results in Figure 7.

A number of papers presented non validated theoretical models and methodologies, and were discarded. The literature reviews were read, but their data were not used for the PP definitions, as we wanted to base our definitions solely on practices used directly in the field. Our motivation for a limitation on observations was that *"no one claimed that this was practiced as written in the books"* [S22]. We surmised that the canonical definitions of PP might not be the same as the definitions actually used in the field.

Concerning the empirical papers, we found that a large number of them are based on laboratory experiments. Short-term experiments are problematic, because it takes time for developers to understand the dynamics of PP and to jell as a team. Evidence seems to show that it takes at least 10 hours of work for a pair to attain a jelled state ([S25], [S28] and [51], [52], [53]). As Williams et al. write [S28]: *"Pairing is significantly more expensive when programmers are first learning the dynamics of pairing."* Therefore, all studies that lasted 10 hours or less were considered laboratory experiments and discarded.

Therefore, the field observation papers category consists mostly of industrial papers, along with a few lengthy academic studies. Six papers involve a context unrelated to the industry, however. These papers typically address the question of how PP could be used to improve learning at the middle school level. These papers were discarded.

This secondary selection process resulted in 36 papers, which presented empirical data in a field setting that could be related to real software development work.

### 3.3. Data Analysis

One of the key features of PP papers is the amount of pairing actually taking place during the study. We found that this amount varies widely, from 10% to 100% of total project effort. This difference in PP activity level might explain the observed difference in PP effectiveness. The dynamics of a team in which less pairing occurs is probably different from the dynamics of a team in which a high degree of pairing occurs. Data analysis has therefore evaluated the amount of pairing reported in the selected papers.

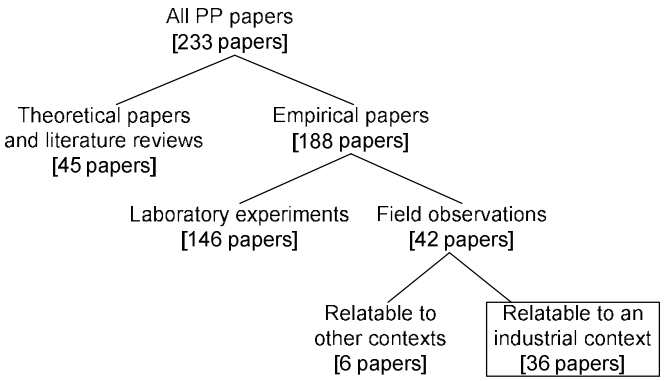


Figure 7. Categorization of the PP papers selected.

# 4. CITATIONS

## 4.1. Pairing Results

Figure 8 presents the level of pairing in the 36 selected studies. We note that for 9 of the studies, the level of pairing is unclear. As Lanubile and Mallardo report [S14]:

*"A threat to the validity of our study is that we could not observe the developers while working, so we cannot exclude [the possibility] that pairs effectively worked as driver/observer rather than splitting the assignment and working individually."*

Pairing levels are not always monitored, and, when they are, the levels are described in vague terms. This makes comparison between studies difficult, as the reported level of pairing varies between 10% [S26, S27] and 100% [S17, S24, S25, S30, S31, S33]. Note that some papers present different analyses based on the same dataset. These papers are presented as children of their parent dataset in Figure 8.

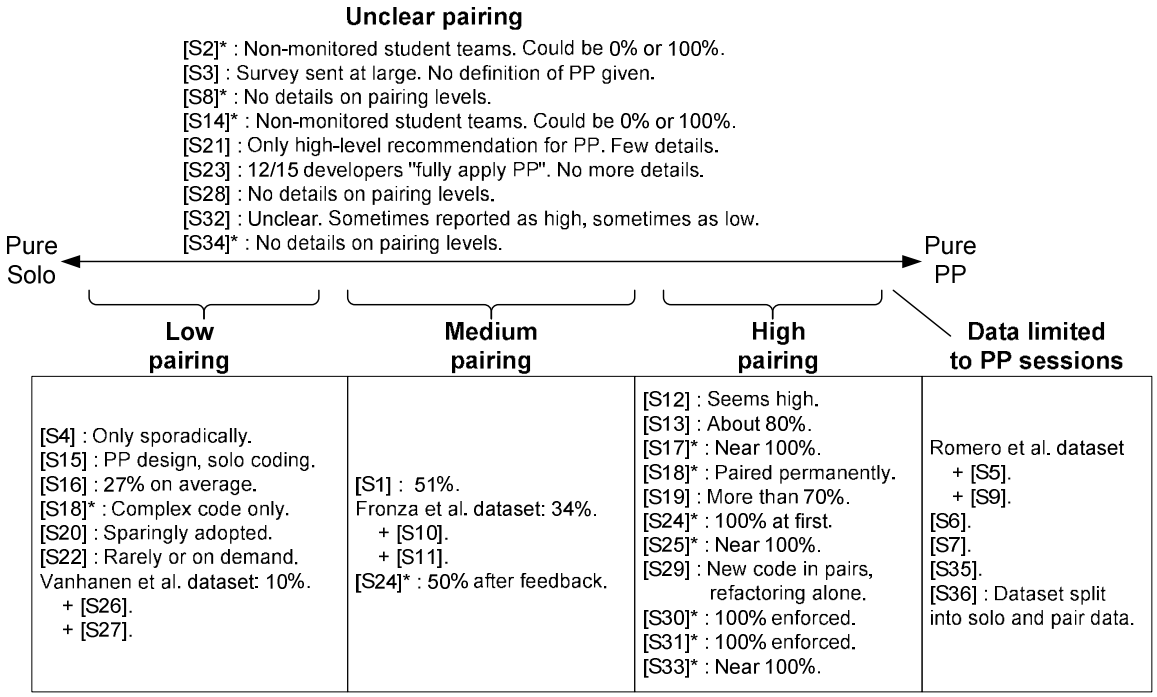


Figure 8. Amount of pairing in the studies selected. Papers marked with an asterisk (*) contain only academic data.

Note, too, that some papers report multiple studies. The [S18] study, for example, reports how student teams adapted the PP practice to their needs. Some teams limited PP use to complex code and so their pairing levels were low, while others paired permanently and so their pairing levels were high. The [S24] study presents two datasets from the same project. The first dataset is limited to the first half of the project, where a 100% pairing level was imposed. The second dataset is limited to the second half of the project, after feedback was received from PP experts. Based on the experts' feedback, management applied more relaxed pairing policies, resulting in a pairing level of about 50% for this second half.

Figure 8 shows that there are slightly more high pairing studies than low pairing ones. This is misleading, however, because many of the high pairing studies were performed in an academic context (papers marked with an asterisk in Figure 8), as shown in Figure 9. There is a key difference between academic and industrial studies: instructors can enforce 100% pairing without consideration of performance or productivity. This might explain why all the studies reporting near 100% pairing are academic, while high pairing industrial studies are less restrictive in terms

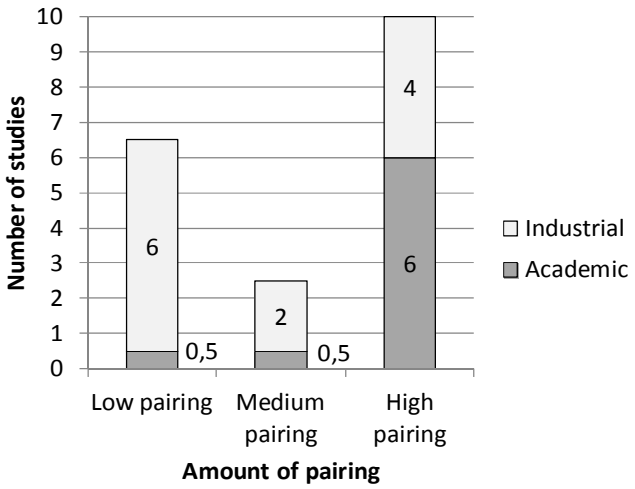


Figure 9. Pairing levels in academic vs. industrial studies.

of pairing.

Four industrial papers, [S12], [S13], [S19] and [S29], report high pairing levels. The [S12] paper reports that coding was performed in pairs first, but became more relaxed as pairing was not always possible because of a lack of an available partner. In [S13], an assessment of pairing levels is reported within the iterations of four case studies. These studies show varying pairing levels as the projects evolved, although the average is kept at a high value of 80%. In [S19], the responses from a questionnaire that was widely distributed among software developers are reported. Of the developers using PP in their daily work, 51% of them work without a partner less than 30% of the time (resulting in an assumed 70% pairing). In [S29], it is reported that all new code is written by pairs of developers, however team size is limited to two developers, and so only one pair is involved.

The high pairing reported in the industrial papers tells us that it is possible to apply a high level of pairing in an industrial context, but also that solo work remains essential. This practice is not in line with the principles of the XP process [34], which requires that all code be written by pairs of developers. The reality of the industrial project does not seem compatible with this principle, at least according to current field observations.

### 4.2. Pairing Levels and the Purposes of Pairing

Still, pairing level is an ambiguous parameter, because the purpose of pairing is not always clear. As shown in Figure 10, pairing efforts can have different purposes, and perhaps only a few of them are dedicated to "true" PP; that is, pairing for shared problem solving in a jelled team.

We theorize that even in a team practicing high level pairing, some incidence of mentoring and

BB may occur. A single PP session could even involve all three identified purposes, in interactions of short duration.

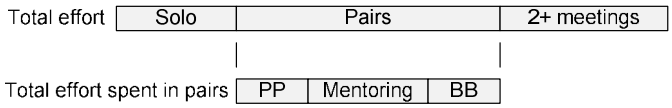


Figure 10. Example of total effort, pairing effort and the various purposes of pairing.

For example, consider the following scenario: Alice and Bob are sitting together in front of one computer to implement a new feature. They have worked together before, and are familiar with each others' quirks and habits. They therefore form a jelled team.

While working on the problem (PP), Alice, at the keyboard, forgot where the configuration file was and asked Bob, who provided the answer (BB). Then, when the work was finished, the solution had to be submitted to the source control tool, with which Bob was not familiar. Alice took control of the keyboard and showed Bob the procedure to follow when working with this tool (Mentoring).

In practice, therefore, what needs to be considered in identifying the purpose of a session is its dominant or overall purpose.

# 5. DISCUSSION

## 5.1. Definition Characteristics discussed in the Literature

Some of the characteristics of the PP definition presented in Table 8 and discussed in the PP literature are presented below.

### 5.1.1. Roles

Evidence seems to show that the driver/navigator metaphor is merely theoretical: Pair programmers do not work at different levels of abstraction, but function rather as a ‘tag team’ [S9], [S35] where both programmers work conjointly at the same problem solving level. However, other roles are also possible that have not been properly studied, based on expertise (domain expert [54], tester [55]...) or personality (introvert/extrovert [35]), for example. Culture and gender factors could also explain some of the dysfunctional nature of pair interactions [35].

Another aspect of role attribution is the purpose of the pairing. Shared problem solving has a different purpose from that of pair mentoring, which has a different purpose from that of BB. The roles within the pair can be defined as tutor and novice, helper and requester, or a tag team [S9].

### 5.1.2. Pair Switching

The definition states that pairs must switch roles every once in a while, to avoid issues like the ‘professional driver’ [35] and the ‘disengaged navigator’ [S35]. Frequent switching between the role of the driver and the role of the navigator is supposed to keep both partners alert. This has not been studied closely in practice, however. For example, it is not clear whether or not an alert team would benefit from imposed switching.

### 5.1.3. Pair Rotation and Rotation Pool

Pair rotation is advisable, and should prevent pair fatigue, where the perception of the two developers becomes too similar [36]. A developer should therefore be able to pair with more than one partner. This means that PP studies based on a team of two developers or on non rotating developers might not constitute true PP work. By removing pair rotation, a critical aspect of PP is removed, which might undermine its potential benefits.

In addition, the pair rotation pool should be limited [35]. If a developer is always pairing with someone new, he/she wastes time determining how to organize the pair’s interactions. In fact, the forming/storming/norming/performing stages of Tuckman's model of group development [56] may apply each time the developer has to pair with someone new. This might explain the long delay required before new pair partners can form a jelled team [S25], [S28].

### 5.1.4. Collocation

Research in distributed PP seems to indicate that remote pairing, while not ideal, is possible [35], [S2]. While distributed PP requires a special tool for communication, it is unclear what communication channels the tool must support. Some research has claimed that a video feed is not necessary [57], [S2], while others maintain that it is [58].

### 5.1.5. Intra-pair Communication

Pairs should communicate regularly in order to benefit from a mutual rubber plant effect. This means that communication channels should be in place to ensure that pair partners can communicate with one another. While the benefits of vocalization have been demonstrated in other contexts [36], it is unclear whether or not there are still benefits when communication is in the form of typed messages.

Communication frequency has been determined informally by W&K as being bounded by a maximum of 45 seconds of silence [35]. Longer silences could mean that a pair partner is inattentive or inactive. Another possible reason for long silences is that the task should not be

performed in pairs. Either the task is too trivial, or the solution is already known by both developers.

### 5.1.6. Pair Schedule

Management should allocate time for pairing sessions. As W&K argue [35], modern flex time schedules mean that developers may not be in the office at the same time. If no time is set aside for pairing, then it can be argued that less pairing will occur.

### 5.1.7. Pair Assessment

Integration of the PP practice into a software development team will involve some adjustment from management. Pair work should not be assessed in the same way as solo work. An implicit assessment of the developers in a team is already made by the pair partners [36]. The practice of PP ensures that the partners implicitly identify the technical expertise and collaborative skills within the team. Management could use the information garnered from this implicit assessment for the performance evaluation of their PP developers, as well as for the formation of efficient pairs.

### 5.1.8. Tasks

There is sufficient evidence to support the idea that PP could be beneficial for tasks other than strictly coding ones [35], [S1, S15]. The use of pairs during the specification, design, and testing phases of software development has already shown some interesting potential benefits.

### 5.1.9. Other Aspects

The common definitions of PP do not mention the use of special tools. A tool is mandatory for distributed PP, although the capabilities this tool must have are still unclear. A tool could also be defined for collocated PP. For example, a tool could be built to identify the pair partners available, to help assess pair partners, to measure the time spent at the keyboard, etc.

Another aspect not covered by the common definitions of PP is synchronicity. This concerns distributed PP, as face-to-face PP is synchronous. The problem is, what is the acceptable delay between questions and answers in a distributed PP setting? Intra-pair communication should occur every 45 seconds, as described previously, but, with modern systems, this would be possible using email. Is PP possible simply with email, or with texting? More research is required to identify the level of synchronicity required.

Finally, the importance of personality in the success or failure of PP has been repeatedly demonstrated [59], [60], [61], [62], [63], [S23]. Coupled with the observation that developers

implicitly assess the expertise and personality of their pair partners [36], the conclusion is that developers might have preferences as to who they partner with. Whether management selects pair partners or allows the selection to happen naturally could affect developers' resistance to the practice.

### 5.2. On Context Variables

The number of context variables deemed important for evaluating PP effectiveness in the works of Gallis et al. and Ally et al., and among our own contributions, is enormous. We listed 38 of them in Table 9! This makes the monitoring of PP studies difficult.

The question, therefore, is which of these 38 context variables are significant? We can also ask whether or not their significance can be determined from existing data. We think that it is not currently possible to do so, because most key variables have not yet been accurately measured in enough studies. Most studies limit the context description to project duration, team size, and team members' experience. Task complexity is sometimes described, in the form of lines of code or use cases implemented. However, the approach used to manage the practice of PP is rarely described. In order to determine the importance of these variables, future PP experiments will need to be carefully monitored.

### 5.3. On Exotic Variants of Pair Programming

While sifting through the literature, we found many exotic variants of PP. This raises an interesting question about whether our current PP approach is the most appropriate. The following list presents some PP variants which are less conventional and could warrant more examination:

- **Tri-programming**: One developer works at a keyboard, another developer watches him work and validates his input, and a third programmer performs whiteboard design [64].
- **Side-by-side programming**: Two developers sit side by side with two computers, two keyboards, and two mice, and both of them have read/write access to the code. They do not work on the same problem, but can see what the other is working on. This approach incites informal BB and PP exchanges, since a developer can easily see when his/her partner is stuck on a problem [65], [66], [67].
- **Side-by-side test-oriented pair**: Using a similar setup as in the previous example, two developers work on the same problem: One developer writes the user story code, while the other developer writes the user story unit tests [55].

- **Simulated partner**: An AI agent simulates the validation and design planning of a human pair partner [68].
- **Seminar mentoring**: A class of student developers shares problem solving approaches to programming issues using group programming [69].
- **Dual programming**: Two developers work on the same feature, but build a solution alone. They then exchange their code and write unit tests for the feature. Their insights can then be merged, in order to obtain the final code and the final test suite [70].

Still more variants can be obtained by playing with the characteristics of the PP definitions presented in Table 8.

### 5.4. On Distributed Pair Programming

There is an extensive body of research on the subject of distributed pair programming, as shown in Figure 11. Figure 11 shows the publication date of all distributed PP studies from the initial selection of 233 papers. The idea of pairing with a distant partner is of interest in conventional PP studies, because it is an occasion to identify the communication channels and interactions required to make PP work.

The question still remains as to whether or not distributed PP is comparable to face-to-face PP. Most of these papers present a specific distributed PP tool or a short laboratory experiment. Only one paper showed enough detail to be retained in our final selection [S2].

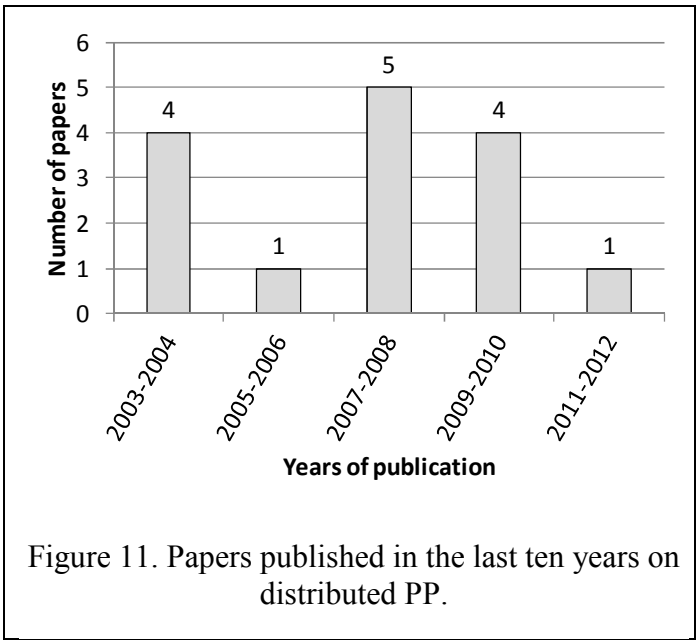


Figure 11. Papers published in the last ten years on distributed PP.

## 5.5. On the Threats to Validity

### 5.5.1. The Selection Process

The main threats to the validity of a literature review are related to the repeatability of the selection process and of the data analysis. The selection process we used closely followed the recognized methodology defined by Kitchenham and her colleagues [21], [71], [72].

The selection was performed by a single researcher. However, a second researcher validated random samples of the work performed by the first researcher. Initial samples showed low inter-rater agreement. As a result, selection was restarted until the samples presented a near-consensus on the selection process. About 200 papers were inspected in this way for sample evaluation out of all 1,362 papers returned by the database searches. The data analysis was performed by two researchers. Interpretations of the data extracted from the selected papers were discussed until a consensus could be reached.

It is our belief, therefore, that the conclusions presented in this paper are representative of the current state of the field.

### 5.5.2. Hannay et al.'s Literature Review

Currently, the best literature review in the field is the extensive meta-analysis on the effectiveness of PP published in 2009 by Hannay, Dyba, and Arisholm [33]. This review selected 18 high quality papers evaluating the impacts of PP on quality, project duration, and overall effort.

A high level of correspondence between Hannay's selection and ours would be expected. However, this is not the case: Only one paper selected by Hannay et al. made it into our selection. We reviewed our selection process and found that all Hannay's papers[5] were selected for our initial package, which contained 233 relevant papers. We found that all the missing Hannay et al. papers were in the laboratory experiment category, shown in Figure 7.

This might explain why the meta-analysis of Hannay et al. reached the mitigated conclusion of ‘it depends’: It compares studies with divergent states of PP. Laboratory experiments are too short for a proper analysis of the effectiveness of PP, as it takes time for a pair to jell [S25, S28]. As one of the studies they selected reports, at the end of a 6.5 hour experiment [73]:

[5] One paper was published before 2000, and was therefore not selected. Another paper could not be found with our search string of "pair programming", or with Hannay's reported search string, and so it was probably selected through a snowball search.

*"It was not possible to avoid learning effects during the experiment: as a matter of fact, significant differences between the outcomes of the runs were detected."*

We still believe that their meta-analysis is the best in the field, mainly because the selected papers incorporate some of the best data available. We have managed to select more field observational studies, 36 papers to be exact, because we were not limited to papers reporting quality, duration, and effort metrics. We only needed data on the definitions of PP used. Arguably, however, 11 of our selected papers were based on work performed in an academic context, and might not be entirely relatable to the industrial context.

At the risk of making an often repeated statement yet again, software engineering needs more field observations relatable to industrial projects in order to arrive at clear conclusions. This is especially true in a domain as complex and ambiguous as PP.

## 6 CONCLUSION

This literature review outlines, once again, the importance of field observations for the evaluation of PP effectiveness. Most of the current laboratory experiments do not replicate the complex interactions seen in the industry. This literature review has presented the key characteristics of the definition of PP, along with the context variables required to determine these characteristics in the field.

Based on previous research and our own contribution, there are 38 context variables that potentially have an impact on PP effectiveness, albeit only 21 of them are both controllable and measurable. This is still too large a number for monitoring a PP study and the most significant of these variables must be identified. One of the issues is that PP was first introduced as a management practice, dedicated to ensure that developers follow good practices in a lightweight process [5]. This practice has escaped the management field, and is now studied for its potential benefits as a software development practice. This implies a change in the focus of the research, from a management perspective interested by resources and schedule, to a software engineering perspective interested by role responsibilities, work methodologies, input requirements and output quality. This evolving change of focus might explain the multidimensionality of PP studies, which must report both management results (e.g. productivity) and software engineering results (e.g. product quality).

As well, many exotic variants of PP have been developed in the last decade. These variants raise the question of the appropriateness for software development of the canonical definition of PP presented in books.

In summary, the benefits of PP, seen as a unified approach, are still unclear. The literature shows that this practice is complex, both in its management within a project (pair rotation, pair switching, pair scheduling, etc.), in the nature of the interactions taking place (shared problem solving, pair mentoring, backup behavior), and in the nature of the practice itself (management or software engineering). What this study demonstrates is that, in order to understand the benefits of PP in software engineering, we must first evaluate its use within the context of jelled teams.

The question then becomes: When two people work together, is it always PP? This "two-person-at-one-computer" definition is very large and covers a whole range of practices which might not be what the researchers originally had in mind. The literature shows that the purpose of having two individuals sit together at one computer can vary widely, from a simple transfer of information to a mutual effort toward a single, common goal. And if the purposes for pairing are different, then the benefits of pairing are likely to be different as well.

**Appendix – The Selected Studies**

[S1] J. Auvinen, R. Back, J. Heidenberg, P. Hirkman, and L. Milovanov, “Software process improvement with agile practices in a large telecom company,” in 7th International Conference on Product-Focused Software Process Improvement (PROFES 2006), 2006, pp. 79–93.

[S2] M. Bandukda and Z. Nasir, “Efficacy of distributed pair programming,” in 2010 International Conference on Information and Emerging Technologies (ICIET), 2010, pp. 1–6.

[S3] A. Begel and N. Nagappan, “Pair programming: What’s in it for me?,” in 2008 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM’08), 2008, pp. 120–128.

[S4] S. Bryant, “Double trouble: mixing qualitative and quantitative methods in the study of extreme programmers,” in 2004 IEEE Symposium on Visual Languages and Human Centric Computing, 2004, pp. 55–61.

[S5] S. Bryant, P. Romero, and B. du Boulay, “The collaborative nature of pair programming,” in 7th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2006), 2006, pp. 53–64.

[S6] J. Chong and T. Hurlbutt, “The social dynamics of pair programming,” in 29th International Conference on Software Engineering (ICSE’07), 2007, pp. 369–378.

[S7] J. Chong and R. Siino, “Interruptions on software teams: A comparison of paired and solo programmers,” in ACM Conference on Computer Supported Cooperative Work (CSCW 2006), 2006, pp. 29–38.

[S8] G. Concas, M. Di Francesco, M. Marchesi, R. Quaresima, and S. Pinna, “Study of the evolution of an agile project featuring a web application using software metrics,” in Product-Focused Software Process Improvement (PROFES 2008), 2008, vol. 5089 LNCS, pp. 386–399.

[S9] S. Freudenberg, P. Romero, and B. du Boulay, “‘Talking the talk’: is intermediate-level conversation the key to the pair programming success story?,” in Agile 2007, 2007, pp. 79–86.

[S10] I. Fronza, A. Sillitti, and G. Succi, “An interpretation of the results of the analysis of pair programming during novices integration in a team,” in 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009), 2009, pp. 225–235.

[S11] I. Fronza, A. Sillitti, G. Succi, and J. Vlasenko, “Analysing the usage of tools in pair programming sessions,” in Lecture Notes in Business Information Processing, 2011, vol. 77 LNBIP, pp. 1–11.

[S12] A. Fruhling and G.-J. De Vreede, “Field experiences with extreme programming: developing an emergency response system,” Journal of Management Information Systems, vol. 22, no. 4, pp. 39–68, 2006.

[S13] H. Hulkko and P. Abrahamsson, “A multiple case study on the impact of pair programming on product quality,” in International Conference on Software Engineering (ICSE 2005), 2005, vol. 2005, pp. 495–504.

[S14] F. Lanubile and T. Mallardo, “Inspecting automated test code: A preliminary study,” in 8th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2007), 2007, vol. 4536 LNCS, pp. 115–122.

[S15] G. Luck, “Subclassing XP: breaking its rules the right way,” in Agile Development Conference (ADC 2004), 2004, pp. 114–119.

[S16] K. M. Lui and K. C. C. Chan, “Software process fusion by combining pair and solo programming,” IET Software, vol. 2, no. 4, pp. 379–390, 2008.

[S17] L. Madeyski, “Impact of pair programming on thoroughness and fault detection effectiveness of unit test suites,” Software Process Improvement and Practice, vol. 13, no. 3, pp. 281–295, 2008.

[S18] M. M. Muller and W. F. Tichy, “Case study: extreme programming in a university environment,” in 23rd International Conference on Software Engineering (ICSE 2001), 2001, pp. 537–544.

[S19] W. Pedrycz, B. Russo, and G. Succi, “A model of job satisfaction for collaborative development processes,” Journal of Systems and Software, vol. 84, no. 5, pp. 739–752, 2011.

[S20] C. Poole and J. W. Huisman, “Using extreme programming in a maintenance environment,” IEEE Software, vol. 18, no. 6, pp. 42–50, 2001.

[S21] D. J. Reifer, “How to get the most out of extreme programming/agile methods,” in Extreme Programming and Agile Methods - XP/Agile Universe 2002, 2002, pp. 185–196.

[S22] C. Schindler, “Agile software development methods and practices in Austrian {IT-industry:} results of an empirical study,” in International Conference on Computational Intelligence for Modelling Control & Automation (CIMCA 2008), 2008, pp. 321–326.

[S23] P. Sfetsos, L. Angelis, and I. Stamelos, “Investigating the extreme programming system - an empirical study,” Empirical Software Engineering, vol. 11, no. 2, pp. 269–301, 2006.

[S24] M. B. Smrtic and G. Grinstein, “A case study in the use of extreme programming in an academic environment,” in 4th Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2004, 2004, pp. 175–182.

[S25] J. Vanhanen and C. Lassenius, “Effects of pair programming at the development team level: An experiment,” in International Symposium on Empirical Software Engineering (ISESE 2005), 2005, pp. 336–345.

[S26] J. Vanhanen and C. Lassenius, “Perceived effects of pair programming in an industrial context,” in 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007), 2007, pp. 211–218.

[S27] J. Vanhanen, C. Lassenius, and M. V Mantyla, “Issues and tactics when adopting pair programming: A longitudinal case study,” in 2nd International Conference on Software Engineering Advances (ICSEA 2007), 2007.

[S28] L. Williams, A. Shukla, and A. I. Anton, “An initial exploration of the relationship between pair programming and Brooks’ law,” in Agile Development Conference (ADC 2004), 2004, pp. 11–20.

[S29] W. A. Wood and W. L. Kleb, “Exploring XP for scientific research,” IEEE Software, vol. 20, no. 3, pp. 30–36, 2003.

[S30] K. Stapel, D. Lübke, and E. Knauss, “Best practices in extreme programming course design,” in Proceedings of the 30th international conference on Software engineering, 2008, pp. 769–776.

[S31] J. Noble, S. Marshall, S. Marshall, and R. Biddle, “Less Extreme Programming,” in Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30, 2004, pp. 217–226.

[S32] F. Grossman, J. Bergin, D. Leip, S. Merritt, and O. Gotel, “One XP experience: introducing agile (XP) software development into a culture that is willing but not ready,” in Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research, 2004, pp. 242–254.

[S33] W. Chigona and M. Pollock, “Pair programming for information systems students new to programming: Students' experiences and teachers' challenges,” in Management of Engineering Technology, 2008. PICMET 2008. Portland International Conference on, 2008, pp. 1587–1594.

[S34] L. Layman, “Changing students’ perceptions: an analysis of the supplementary benefits of collaborative software development,” in 19th Conference on Software Engineering Education & Training (CSEET 2006), 2006, 8 pp.

[S35] L. Plonka, H. Sharp, and J. van der Linden, “Disengagement in pair programming: Does it matter?,” in Software Engineering (ICSE), 2012 34th International Conference on, 2012, pp. 496–506.

[S36] A. Sillitti, G. Succi, and J. Vlasenko, “Understanding the impact of Pair Programming on developers attention: A case study on a large industrial experimentation,” in Software Engineering (ICSE), 2012 34th International Conference on, 2012, pp. 1094–1101.

**Acknowledgment**

This work was supported in part by the NSERC grant A-0141.

# ANNEX 2: PERFORMING SYSTEMATIC LITERATURE REVIEWS WITH NOVICES: AN ITERATIVE APPROACH

*AUTHORS: MATHIEU LAVALLÉE, PIERRE N. ROBILLARD, REZA MIRSALARI.*

*This article was submitted to IST journal on March 14, 2013.*

*My contribution to this article :*

*I gathered, compiled and analyzed the data needed for pair programming practices as one of the cases which studied during this research in order to demonstrate the efficiency of newly improved method (iSR).*

My specific contribution to this paper is marginal.

**Abstract**

**Context**: Systematic Literature reviews are performed to get a deeper understanding of a specific research domain. However, current literature review processes require domain experts if they are to be successful. But what if a domain expert is not available? What about emerging fields in which no domain expert exist? **Objective**: This paper presents an iterative method of conducting systematic literature reviews designed to address some of the current problems faced by novice reviewers. **Methodology**: Our improved systematic literature review method is based on the present state of the literature on evidence-based software engineering (EBSE). It also benefits from our own experience performing systematic literature reviews with novices. **Conclusions**: The iterative method enables novices in both reviewing methods and domain knowledge to successfully perform a systematic literature review. It is easily scalable, and can be tailored to the available resources. It also addresses some of the most pressing issues of current literature reviews, among them the difficulty of transforming novices into experts.

**Keywords**

Systematic literature review, SLR, scoping review, mapping study, EBSE, review process.

## 1. Introduction

Performing a systematic literature review (SLR) is akin to clearing a path in the middle of a jungle of publications. An expert is a valuable resource when trying to objectively determine

where the path should go, which area it should cover, what structure can be seen within the chaos. It takes an expert to find the relevant studies, compile the important conclusions, analyze the key data, and synthesize the state of knowledge of the field. However, recent advances in the systematic literature review process have pushed this art into the realm of science, through the definition of clear procedures and simplified methodologies [10], [21], [41], [74], [75] to the point where the task is simple enough today to be undertaken by novices or students [50], [76], [77], [78], [79], [80]. Despite the use of novices, expert review is still recommended for the formulation of research questions and to plan the selection procedure [10], [41], [45], [81]. However, the recent advances in understanding SLR processes can lead to its use as a tool to introduce novices to new domains or to help advanced students to become experts in their field.

This paper presents an iterative systematic literature reviews approach designed for those with little expertise in the domain as well as on the review process. The approach presented is based on the current state of the literature and our own experience teaching students how to perform SLRs. This paper aims to answer the following two questions:

- RQ1: What are the difficulties and the risks of performing systematic literature reviews with novices?
- RQ2: How can we mitigate these risks?

This paper reviews the current systematic literature review methods, pointing out where the problems are and how to address them, based on four case studies from the classroom.

### 1.1 What are Systematic Literature Reviews?

Systematic Literature Reviews (SLR) are distinguishable from common literature reviews in the following three ways:

- They follow a clearly defined methodology.
- They aim to synthesize the answers to specific research questions from the whole body of relevant literature.
- They have a repeatable methodology. Anyone using the same methodology and answering the same questions should reach the same conclusions.

SLRs are typically built to answer a question based on the population/intervention/outcome paradigm. For example, a possible SLR question might be: *"What are the effects of the use of Pair Programming (intervention) by collocated professional software developers (population) on*

*code quality metrics (outcome)?"* The conclusions of SLRs are typically obtained through the analysis and synthesis of the conclusions of a handful of high quality studies.

SLR methods were originally developed for software engineering by Biolchini et al. [10] and Kitchenham et al. [82]. The explicit tasks of the SLR process are presented later in Table 10, and are listed below with the corresponding definition from Biolchini et al. [10]:

1. Question formulation: *"In this section, the research objectives must be clearly defined."*
2. Source selection: *"The objective of this section is to select the sources where searches for primary studies will be executed."*
3. Studies selection: *"Once the sources are defined, it is necessary to describe the process and the criteria for studies selection and evaluation."* This also includes the selection execution task, whose aim is to *"to register the primary studies selection process, reporting the obtained studies and the results of their evaluation."*
4. Information extraction: *"Once primary studies are selected, the extraction of relevant information begins. In this protocol section, extraction criteria and results are described."*
5. Results summarization: *"This systematic review protocol section aims to present the data resulting from the selected studies."*

The SLR process also defines tasks specific for a gate-based process, namely inspection and evaluation tasks. The two SLR tasks not shown in Table 10 are presented below with their corresponding definitions from Biolchini et al. [10]:

- Planning evaluation: *"It is necessary to evaluate the planned review. A way to perform such evaluation is to ask experts to review the protocol. Another way to evaluate the planning is to test the protocol execution."* The goal of the iSR process is to forgo the use of domain experts, through the use of multiple refinement iterations, which can act as execution protocol tests.
- Execution evaluation: *"During the execution phase, several problems may occur to due web search engines limitations."* There is no equivalent in the iSR process because this problem is regularly evaluated and the risk is mitigated through the use of iterations.

**1.2. What is Evidence-Based Software Engineering?**

Evidence-based Software Engineering (EBSE) is a form of literature review typically used to select a technology for solving a specific problem. It looks for engineering solutions and

typically covers all types of literature, from peer-reviewed papers to publicity tracts. EBSE in software engineering has been advanced by Kitchenham et al. [50], Baldassarre et al. [81], Jorgensen et al. [78], Rainer et al. [75], [77], [80], and Dyba et al. [21]. The explicit tasks of the EBSE process are presented in Table 10, and are listed below with their corresponding definitions from Kitchenham et al. [10], [21], [41], [50], [74], [75], [76]:

1. Question formulation: *"Convert a problem or information need into an answerable question."*
2. Literature search: *"Search the literature for the best available evidence to answer the question."*
3. Relevance quality and research quality evaluation: *"Critically appraise the evidence in the literature."*
4. Integration with expertise and decision-making: *"Integrate the evidence with practical experience and the circumstances to make decisions about practice."*
5. Process improvement: *"Evaluate the performance in steps 1-4 and seek ways to improve it."*

**1.3. What are Scoping Reviews (Mapping Studies)?**

Scoping reviews, also called mapping studies, are similar to SLRs; they use clearly defined and repeatable methodologies. Their focus is however to map the studies pertaining to a particular topic, instead of answering a specific research question.

Scoping reviews follow a procedure similar to the one used by SLRs, except at the analysis and synthesis stages. Data extraction is limited to a categorization of the papers, and the synthesis is typically limited to a presentation of the categorization scheme, through graphical maps or bubble charts. The quality of the selected studies is rarely evaluated. This limited analysis, synthesis, and quality evaluation enables the scoping reviewers to select a much larger body of data than is possible in a systematic review. Scoping reviews are sometimes used to evaluate a domain before a more extensive literature review is undertaken [41]. Scoping review procedures were originally developed for software engineering by Kitchenham et al. [50] and Petersen et al. [83].

There is no clear process for the execution of scoping review. Previous works by Kitchenham et al. [50] suggest following the EBSE process.

**2. Building the Iterative Literature Review Approach**

Traditional SLR methodologies are described by a waterfall-like review processes. While these methodologies sometimes mention the need to reiterate on some aspects, as shown in section 4.1, they do not make this their core point. Our perception is that performing a SLR is similar to performing a software development project: It is a knowledge-accruing endeavor, which benefits from constant redefinition.

The iterative Systematic Review (iSR) approach presented in this paper is based on Biolchini et al.'s SLR method [10] and Kitchenham et al.'s EBSE method [21] and is derived from the lessons learned from four case studies. Table 10 shows the tasks used by these three literature review methods. The tasks for the iSR are described and justified later in Section 3.

The iSR approach renamed two tasks taken from the traditional SLR method: Source selection and Results summarization. Source selection seemed too limited a term for a task mainly dedicated to building a search string. Results summarization does not outline the main difficulty of the task, which is the building of new knowledge through the synthesis of the extracted information.

Table 10 shows that the iSR method adds an initial task for the planning activity. The needs to plan the review activities are implicit for all methods. The iSR method makes the planning step explicit because the iterative phases may change the initially planned activities.

**2.1. On Phases**

The lifecycle of a systematic literature review can be represented by five phases, each of which being identified with a milestone. The five phases are presented in Fig. 1 with the tasks of the review processes. The objective of the domain definition phase (A) is to define the scope of the review. The phase ends when reviewers define the initial research question, determine some of the boundaries of the research domain, found some relevant keywords, built some basic search strings and have identified a few relevant papers.

Table 10. List of the explicit tasks required for each of the review method: the traditional SLR method, the EBSE , method, and the proposed iterative systematic review method (iSR).

| **SLR** [10] | **EBSE** [21] | **iSR** |
|---|---|---|
| | | 1. Planning and training |
| 1. Question formulation | 1. Question formulation | 2. Question formulation |
| 2. Source selection | 2. Literature search | 3. Search strategy |
| 3. Studies selection | | 4. Selection process (relevance quality) |
| | 3. Relevance quality and | |

| **SLR** [10] | **EBSE** [21] | **iSR** |
|---|---|---|
| | research quality evaluation | 5. Strength of the evidence (research quality) |
| 4. Information extraction | 4. Integration with expertise and decision-making | 6. Analysis |
| 5. Result analysis | | 7. Synthesis |
| | 5. Process improvement | 8. Process monitoring |

The objective of the relevant studies phase (B) is to estimate the size of the domain under review. The phase ends when the reviewers have estimated the number of relevant studies and identified most of them. At this stage, the research questions can be revisited so the amount of relevant studies can be managed with the resources available.

The objective of the selected studies phase (C) is to build the list of selected papers of the review. The phase ends when the body of selected papers has stabilized. This is done by tuning the research questions, the source databases, the search strings and the selection process. New studies can be added afterwards, for example with a snowball search, but the main body of papers should be known at this point. The effort needed to do the subsequent phases is also estimated.

The objective of the extracted information phase (D) is to collect the evidence from the body of selected papers. This phase ends when most of the relevant information has been extracted from the selected papers. Adjustments of the synthesis might require the reviewers to extract some more details, but at the end of this phase, most of the relevant information should be captured in the extraction forms.

The objective of the new knowledge phase (E) is to create new conclusions based on the evidence extracted in the selected papers. The phase ends with the answers to the research questions, based on the synthesis of the evidence. Recommendations for futures reviews are also presented based on the feedback obtained throughout the process.

Figure 12 shows how the five phases of a systematic review lifecycle are related through the iterations to the eight tasks of the iSR process.

**2.2. On Iterations**

The iSR approach is iterative because after any planned iteration the tasks to fulfill the required milestone can be reconsidered. The number of iterations can vary from one project to the other. For illustrative purposes, in Figure 12 each phase has been arbitrarily divided into two iterations. Iterations are defined to mitigate the risk associated to the achievement of the phase's

milestone. Some tasks must be repeated for various reasons: For example, a protocol might not have been appropriately followed, or results might be of poor quality.

The following describes a typical iteration. For example, the iteration #5 presented in Figure 12 may involve each of the eight tasks at various levels of effort and in various sequences. The objective of iteration #5's phase, the selected studies phase, is to list the relevant papers. Therefore, iteration #5 is planned to make sure that the novice reviewers know how to make relevant selections. After the Planning and training task (1), some Selection process task (4) is performed on only few (for example five papers) of the numerous relevant papers obtained from the previous iteration. The three following tasks (Strength of evidence, Analysis and Synthesis) are performed on these five papers. The Question formulation is revisited (2) and the Search strategy (3) is validated. These activities have been Monitored (8) and based on these data the effort and the number of papers selected will be estimated for the next iterations. Novice reviewers are now aware of the process for the selected studies phase and are likely to be more efficient in performing iteration #6.

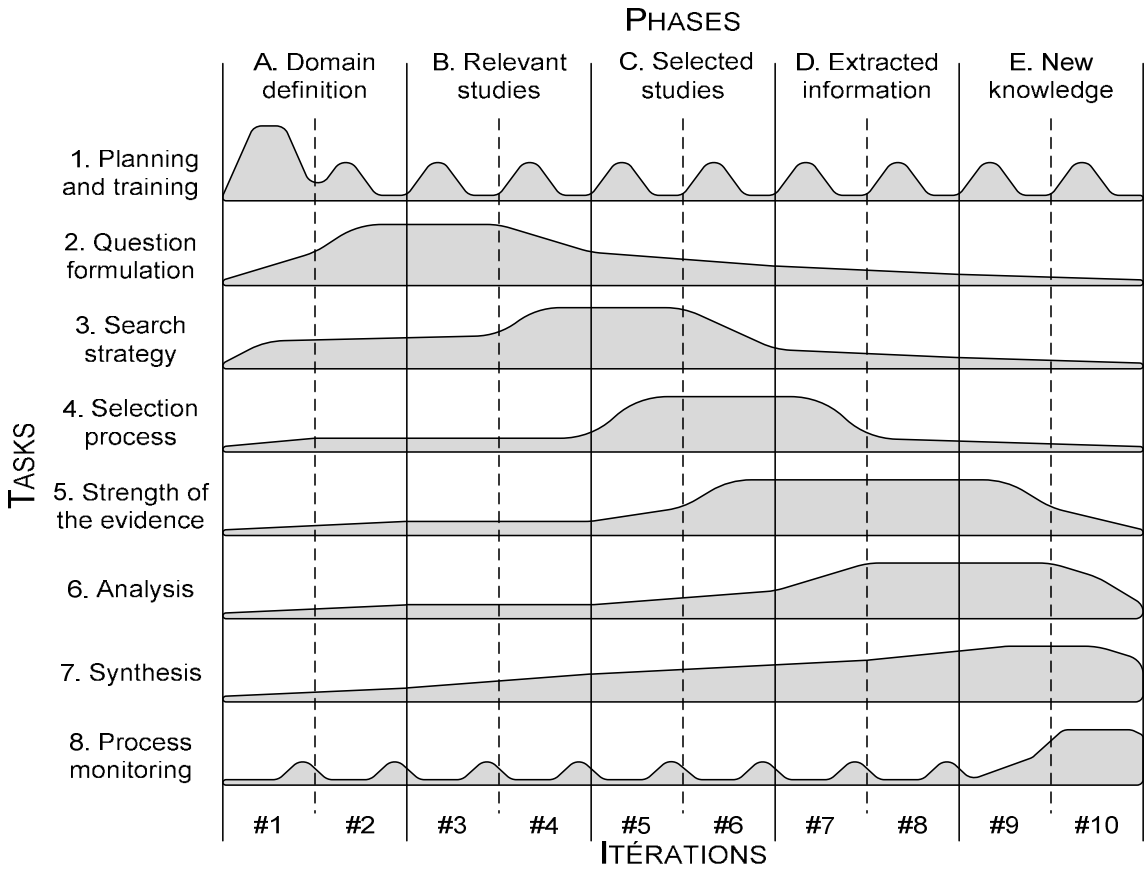


Figure 12. Artist's view of effort expanded during a typical literature review performed using the iSR approach.

This paper proposes a formal planned iteration approach because we found that the SLR reviews performed by novices required two level of understanding: Understanding the SLR process itself and understanding the knowledge domain. The planned iteration approach enables a progressive learning from the two levels of understanding. For example, for a given two iterations phase, the first iteration validates the learning of the SLR process while the second iteration actually achieves the phase's milestone. As far as we know, no other reviewers have explicitly proposed an iterative approach to performing literature reviews. Nevertheless, exercises, dry runs, and repetitions are recommended by many authors for many tasks. The SLR process defined by Biolchini et al. [10] does define phases, and mention the importance of iterations. But the iteration recommendation remains limited to adjustment between the task plan and the task execution: *"Many activities are initiated during the protocol development stage, and refined when the review proper takes place"* [10].

### 2.2.1. Repetition of the planning task in the literature

Brereton et al. [84] make this point: *"The students found the SLR process difficult to understand, with the consequence that the protocol took longer than expected to develop and required many changes and revisions."* This shows that iterating on the review planning stage of the process can be essential, especially when novice reviewers are involved.

Jorgensen et al. [78] do not mention iterations, but their review method used a two step approach; the first delivered an initial definition of the problem, and the second delivered a complete report.

We found that planning should include dry runs of the activities when novice reviewers are involved. The first time an activity is performed the novice reviewers typically focus on the methodology to the detriment of producing good content. As the methodology is mastered, novice reviewers can more easily focus on producing good results, as Bloom's taxonomy suggests [85].

### 2.2.2. Repetition of the question formulation task in the literature

Kitchenham et al. [50] present a case where a reviewer had to revise the topic targeted, because the initial definition covered too wide a spectrum. This required a reboot of the review, as the number of articles retained was initially too large to be properly analyzed.

Oates and Capper [86], following Kitchenham's recommendations, modified the EBSE process, so that a very generic question is posed first and then it is refined as the subject becomes better understood. Consequently, they recommend an iterative approach for the question formulation activity. According to Oates and Capper, *"Experienced researchers may be able to follow a top-down, linear approach: planning, conducting, and then reporting the review. However, novices need iterative and incremental cycles of planning and conducting the review."* They note that the Evidence for Policy and Practice Information and Co-ordination Centre (EPPI-Centre) also recommends an iterative approach for certain reviews. However, Oates and Capper's iterative approach remains limited to the question formulation and search strategy stages. They do not extend its use to other review activities.

The need to iterate on question formulation was confirmed during our own systematic reviews. A failure to review the research questions was the main reason for our Fall 2011 review failure, as will be explain later.

### 2.2.3. Repetition of the search strategy task in the literature

Oates and Capper [86] recommend updating the search strategy, along with the question, as long as all relevant articles are found or a manageable number of articles are selected.

We found that building a good search string is an art best mastered through practice. Many exercises with constant feedback are required before good search strings are created. Reviewers also reported that the term impact analysis approach is easy to use and provide good feedback on the quality of a search string.

**2.2.4. Repetition of the selection process task in the literature**

Rainer and Beecham [75] have designed a process loop that includes the search strategy and the selection process. Results are assessed at the end of each loop, and the search strategy and selection process is adapted accordingly.

**2.2.5. Repetition of the strength of the evidence task in the literature**

Many authors consider that evaluating the quality of research is difficult [50], [71], [77] [84], [86]. Our review confirmed this opinion, as we found that performing a good research quality evaluation requires a preliminary reading of many papers in the field under study. It also requires a keen knowledge of what to look for in a study paper, something that novice reviewers, and even some expert reviewers, might not have.

**2.2.6. Repetition of the analysis task in the literature**

Baldassarre et al. [81] recommend test runs with feedback for the extraction procedure. As one student noted, *"I found the forms easier to understand after [working through] a few examples."*

Strauss [[74], p19] shows that qualitative data collection and codification should follow an iterative and incremental approach. The data collected, the code used to qualify the data, and the memos used to organize the codes must be constantly refined as the selected articles become better understood.

We found that the first extractions are generally of poor quality. They tend to improve as the fields to complete become better known and their purposes become clearer. Sharing extraction forms between novices also helped them understand what is expected.

**2.2.7. Repetition of the synthesis task in the literature**

Our reviews show that synthesis benefits a great deal from an iterative approach. To find some structure in the large amount of qualitative data requires many attempts. Each attempt might also call for a different view of the data, which might in turn require a revision of the

extraction forms. On many occasions, we realized that the data extracted did not support some interesting synthesis that had emerged, which required us to return to the source paper to extract the missing information.

**2.2.9. Lessons learned from our reviews**

Repetitions are not iterations. An emerging problem can rarely be limited to a single task; it is usually caused by multiple causes spanning different tasks. Therefore, repeating a single task will not solve the problem. An iterative approach based on phases will not repeat a single task, but a set of tasks.

For example, a common issue observed during the domain definition phase is related to the difficulty of setting the domain's boundaries. The iterations should include some search string tests performed on a subset of the databases, in order to find a handful of relevant papers with interesting evidence. These pilot runs help the readjustment of the research questions.

The iterations for the relevant studies phase are to perform a fine tuning of the research question and search strings so the body of literature targeted is coherent with the available resources. We found that the iterations should therefore test many search strings, along with some relevance quality and research quality evaluations, in order to estimate the amount of high-quality papers in the domain.

The iterations of the selected studies phase are dedicated to adjusting the search string and selection process in order to determine a stable body of literature. We found that the inclusion and exclusion criteria need to be adjusted many times as the structure of the domain became apparent.

The iterations of the extracted information phase are dedicated in determining which study details are necessary or can be ignored for the synthesis. The structure of the evidence can require changes in the extraction form, but also on the search string and the body of selected papers. We found that some subtle details of the field only became apparent when the full texts were read.

The iterations of the new knowledge phase are required to synchronize the extraction form with the information needed for the synthesis. We found that initial synthesis effort tend to be naïve and to only touch the surface of the domain. To obtain a deep understanding of the domain required reviewers to observe the studies from multiple points of view and to sometimes adjust the extraction form.

### 2.3. Validating the iSR Approach

Four literature reviews on four different domains were performed in order to build and validate the iSR approach. All reviews were performed with graduate students within a graduate course at our Engineering School. The first review, performed during the Fall 2010 semester was composed of a team of five students, who were both domain novices and novice reviewers. The domain targeted was the impact of software process improvement on developers. The review was successful and the results were published [87]. However, the review was not flawless; a number of problems emerged, mostly at the analysis and synthesis stages. These problematic tasks had to be redone with expert reviewers and are discussed later on.

The second review was performed during the Fall 2011 semester, using a team of fourteen graduate students, who were also domain novices and novice reviewers. The domain targeted was the use of Web-based tools during software development. The review failed; mostly because the size of the body of research on distributed (or global) software development was underestimated. The results did not provide a good synthesis of the domain, since almost all the selected papers had less to do with Web-based tool use than with the management of distant teams. Unfortunately, this review also suffered analysis and synthesis problems and the results were found to be too biased to be pursued.

The problems found in the Fall 2010 and Fall 2011 reviews motivated the construction of an iterative approach; the iSR method. To validate this new method, a third review was conducted during the summer of 2012. The objective of this review was to perform a synthesis of the various definitions of Pair-programming practices reported in software engineering studies. The method used two graduate student reviewers. The reviewers were familiar with traditional literature reviews, but not with the iSR approach. They were domain novices, as they never participated in Pair Programming sessions, and had never researched the domain before. The review was successful, and through many iterations managed to produce a good summary of the domain targeted. The review results lead to a paper submission.

The fourth review was performed during the Fall 2012 semester, using three graduate students, who were both domain novices and novice reviewers. The domain targeted was the research of information on the web by software developers. The review demonstrated the usefulness of an iterative approach, as the research questions had to be adjusted multiple times

during the review to account for the state of the domain. The review results have been integrated into a paper.

## 3. Description of the iSR Approach

This section presents the eight tasks comprising the iSR approach. These tasks are derived from recommendations from the literature and from lessons learned from our successful and failed reviews.

### 3.1. Planning and Training

This task has two objectives: First, to plan the review work, and second, to plan training sessions and tutorials so the novice reviewers have the technical knowledge required to perform a literature review. A number of concepts must be introduced before the review can start, such as those underlying SLRs, the review process in general, the use of publication databases, etc.

#### 3.1.1. Review planning in the literature

The recommendation is that the following topics be presented to novice reviewers at some point during the review process:

- SLRs, scoping reviews, and EBSE, explaining their use according to the focus of the research [50], [76], [77], [78], [79];
- Literature review planning [76], including a planning tutorial [50], [76], [77], [78], [79];
- Scientific method [78], an introduction to help reviewers understand what makes a high quality paper;
- Approaches to empirical study [78], [79], [80];
- Common statistical calculations [78], [79], [80], [45], [81], [82], [83], [84], [87] a brief overview;
- Common research biases [78], an introduction.

On the teaching of statistics, Brereton et al. [84] opines: *"Some statistical knowledge is needed and we found that training in basic statistical techniques, such as proportions, confidence intervals, and significance levels, was required."*

#### 3.1.2. Lessons learned for the review planning task

We recommend that all documentation submitted to the reviewers be carefully written, as the repeatability of the process requires clarity and understandability. Any ambiguous statement can result in divergent results, which are detrimental to repeatability. The use of an iterative

approach resolves part of this problem, by offering opportunities to correct any misunderstanding.

The seminar presentations, tutorial, and exercises should follow the first steps of Bloom's taxonomy of educational objectives [85], in order to progressively introduce the concepts. The three following levels of Bloom's taxonomy should be especially considered:

1. **Remember**: Ask students to follow instructions by rote;
2. **Understand**: Ask students to provide a feedback on the appropriateness of the instructions given;
3. **Apply**: Ask students to tailor the instructions to the context at hand.

Because there are no domain experts available, the reviewers themselves are the best proxy. In these circumstances, Bloom's taxonomy should also be considered for teaching reviewers to perform self-evaluations. During the systematic review process, reviewers will need to differentiate strong search strings from weak ones, high-quality studies from biased ones, etc. These quality evaluations require a high level of understanding, something which cannot be achieved with the rote application of a method.

It was also found, as Brereton [84] underlined earlier, that teaching basic statistical concepts is required. We had used some inter-rater evaluation metrics in our reviews (Cronbach's Alpha and Fleiss' Kappa), and many reviewers complained that insufficient resources, in terms of time and exercises, were devoted to their explanation.

The review on Pair Programming showed the importance of reviewing the work plan. At about the midpoint of the iSR review process, it became apparent that there was no clear definition of Pair Programming in the domain literature. Consequently, the reviewers had to go back and read the references most often cited in the domain, in order to understand how Pair Programming was typically defined. This knowledge requirement was unforeseen at the beginning of the review. It only became clear as the reviewers gradually improved their view of the domain. The review planning had to be revisited to account for this problem.

As shown in Figure 12, this task is needed for all iterations in order to mitigate the problem of exceeding the resources allowed for the iteration and keep the whole process on schedule. In the case studies, we used one week iterations, and the work of the next week was adjusted based on the results of the previous week.

### 3.1.3. Problems to avoid during planning and training tasks

The main problem for this task occurs if the reviewers do not master the concepts and techniques required to perform a systematic review. For example, the review's conclusions can be dubious if the reviewers cannot evaluate the strength of the evidence, and distinguish between a high quality paper and a low quality one.

To mitigate this problem, we recommend that not all required knowledge be transmitted at once to the novice reviewers. For example, the importance of distinguishing high quality papers from lower quality ones does not emerge until a number of potential papers have been selected. Tutorials on the scientific method can therefore be delayed until the concepts are needed. This ensures that previous concepts are mastered, per Bloom's taxonomy levels, before new concepts are introduced.

Another problem is related to deadline pressure. It is tempting to cut corners in order to finish the review process. Lessons learned include the remark that it is preferable to have an unfinished review with good quality data. An unfinished review can always be pursued later on by another team, while a finished but botched review will need to be redone. As presented in section 1.1, a systematic review needs to be objective and repeatable. Botched tasks tend to be biased and not detailed enough to be reproducible.

### 3.2. Question Formulation

The aim of this task is to define research questions and review objectives. Why are we performing this literature review? What do we want to know?

Traditional SLRs require that the questions and hypotheses be defined prior to the research being conducted, in order to avoid the introduction of confirmation and publication biases. By definition, iSR iterative approach allows adjustments to be made to the research questions, and even encourages it, as the evidence will show. We can draw an analogy between research questions for SLRs and requirements for software development: Theoretically, requirements are firmly established at the beginning of a project. This is rarely the case in practice.

Software development requirements evolve as developers better understand the needs of the client and the challenges at hand. They are then adjusted to reflect these changes in perception. Similarly, as we gain a better understanding of the domain under study, our perception of it changes, and, very often, the research questions need to be reformulated. The

initial questions may reflect a somewhat naïve perception of the domain, which might not be in line with reality.

### 3.2.1. Question formulation in the literature

The formulation of research question can be initiated with a very generic question, such as *"What has already been written on subject X?"* [86]. More targeted questions can be introduced as the domain becomes better understood [41].

Question formulation has been found by many authors to be very difficult a task [76], [78], [80], [86]. According to Rainer and Beecham, writing a research question is easy, but writing a good one is hard [77]. They suggest a four-step procedure for writing good research questions [75]:

1. Identify the problem;
2. Write down the relevant definitions required to understand the problem;
3. Write down the assumptions made in relation to the question;
4. Identify the preconceived ideas (hypotheses) you have on the subject.

Following this procedure should provide a solid framework for the research questions, and so ensure a clearer interpretation.

In the opinion of Petticrew and Roberts [[41], p189], for SLR question formulation,

*"A wide range of questions may need to be considered by reviewers of social interventions: not just 'what works,' but why it worked; how was it implemented or delivered; if it did not work in some settings, why not; and what is known of the context within which it was delivered. A review, may, for example include a primary question about effectiveness and a range of further questions about process and implementation factors."*

For EBSE question formulation, Jorgensen et al. provide the following template: *"What is the effect of X, for organizations/developers of type Y, in situations of type Z?"* [78]. Rainer et al. provide a full example of an EBSE research question [75]:

*"Does the use of an object-oriented language (the intervention) produce better quality code (outcome) compared to a structure programming language (comparison/baseline) when used by experienced programmers (uses of the technologies) when developing a web-database site in a short time-frame with a small development team (situation)."*

The SLR and EBSE questions broadly follow the population-intervention-outcome approach to question formulation. Scoping reviews, however, take a more generic approach. For

these kinds of studies, Kitchenham et al. recommend a question like: *"What do we know empirically about topic X?"* [50].

Calibrating the research question is essential, in order to obtain a manageable, yet significant, body of literature. A mapping study based on a tiny handful of papers is not useful because the map can only be fragmentary. On the other hand, an SLR based on hundreds of papers is not pragmatic since it could take years to analyze and synthesize.

### 3.2.2. Lessons learned on question formulation

The question formulation task suffers from a paradox: The domain must be well understood in order to produce good research questions. But, to understand the domain, we must first perform the research, which requires us to have good research questions.

The previous reviews show that the initial research questions need to be adjusted as the domain becomes better known. The main cause of the Fall 2011 failed review was the fact that the question was unsuited for the current state of knowledge of the domain. This domain, of Web-based development tools, has been monopolized by distributed development initiatives, and the research questions should have reflected that fact, either by limiting the population to collocated development tools, or by clarifying that the paper should focus on tool use, and not management practice.

We also found that redefining the research questions proved very useful in the review of Pair Programming (PP) studies. The initial goal was to map the various definitions of PP, but we found during the review process that the definitions were often unclear and incomplete. Consequently, we adjusted the research questions in the following iteration. The final questions were aimed at determining the context of PP research, the available definitions of PP, and the elements required for a full definition of the practice.

### 3.2.3. Problems to avoid during question formulation tasks

The research question determines the scope of the review. As mentioned earlier, the main problem is to define research questions for which no relevant papers can be found, or to define research questions which return hundreds of relevant papers. The scope of the review should be calibrated to the resources available and to the size of the domain.

## 3.3. Search Strategy

This task comprises a procedure for initially retrieving the papers. The procedure sets out the mechanical steps of the selection process, which are performed automatically by the search

engines used. They defined the journals and conferences that will be targeted, the databases that will be searched, whether or not a "snowball" search[6] will be performed, whether or not "grey" literature[7] will be searched, etc.

### 3.3.1. Search strategy in the literature

The construction of a search string often use the structure "population AND intervention AND outcome" [81], [84]. Reviewers build multiple search strings, typically one per article database or search engine used. Building a good search string involves the same paradox as building a good research question. In Brereton's words [76]:

*"It is quite a challenge, especially when you are not an expert in the topic of study. Of course the data should come from the studies, but it is hard to establish the best search strings until you are familiar with the topic."*

The challenge of building a good search string is compounded by the fact that database search engines do not use a standardized language. Some functionalities offered by one database are not offered by another, and vice-versa. To make matters worse, some functionalities are either bugged or poorly defined, or both.

Rainer and Beecham present a short procedure for the construction of a search string [75]:

1. Identify the sources (databases, etc.) that will be searched;
2. Build the search string;
3. Identify inclusion and exclusion criteria;
4. Execute the searches, taking note of the results;
5. Revise the results obtained;
6. Revise the search string and return to step 2.

It is interesting to note that Rainer and Beecham's search string building procedure is iterative. An iterative approach is also supported by Oates and Capper, who state that *"the search strategy is likely to be adjusted as the results are inspected and the research question evolves"* [86].

Note that Rainer and Beecham's procedure includes the definition of inclusion and exclusion criteria. In the iSR approach, this crucial task is defined in the selection process (section 3.4). We

[6] A "snowball" search refers to the practice of reading the reference list of selected papers in the search for additional relevant papers.
[7] Grey literature refers to unpublished papers, or paper published in non-peer reviewed papers. They mostly consist of technical reports, for which the quality can vary from excellent to mediocre.

distinguish between building a search string and defining inclusion and exclusion criteria because the two tasks require different skill sets and must be evaluated differently.

Building the search string is essentially a mechanical task, which requires skills on operating the various databases' search engines. The evaluation of the search string can also be mechanical, as shown in the following section.

Defining inclusion and exclusion criteria is much fuzzier, and requires good judgment and decision-making skills. Quality evaluation of inclusion and exclusion criteria also requires more scrutiny than with a search string.

**3.3.2. Lessons learned on search strategy**

We realized the importance of validating the search string. One of our reviewers developed a method for search string validation called "term impact analysis". This approach consists of testing the search string with and without each term, in order to measure its impact on the results. A term with no impact on the results can be safely discarded. A term responsible for 75% or more of the results might be too generic and could be refined. Table 11shows the partial results of a term impact analysis for one of our case study reviews. This methodology could resolve the problem stated by Rainer et al., which is that *"students provided poor explanations in their reports of how their searches were conducted"* [80].

Another problem of the research strategy is that reviewers tend to perform a typical Google like search. They are looking for the one perfect answer to the research question, instead of looking for all the possible answers. Reviewers must remember that most articles do not provide direct answers to their research questions. Most of the evidence found in reviews provides answers only indirectly.

There were also some technical problems inherent in building search strings. The wildcard characters (?, *) are not always used in accordance with search engine capabilities. Many search strings were found to have orthographical errors in the terms, resulting in fewer results than expected. Some search strings used subsumed terms and were therefore useless, like the search string of "software" OR "software engineering". Also, some search strings used hyper generic terms along with hyper specific ones, resulting in an unbalanced string, like "software" OR "test-driven development".

Most of these problems can be detected by a validation technique like term impact analysis, shown in Table 11.

We also found that a pilot approach works well for the search string. A small sample of the results of a search string can be applied to the selection, analysis and synthesis tasks. This can help reviewer spot relevant and irrelevant keywords which were not included in the initial search string.

### 3.3.3. Problems to avoid during search strategy tasks

The main problem for this task is related to the identification of adequate keywords. The building of a search string requires a minimal knowledge of the domain in order to identify relevant keywords for the research.

We found that some of these keywords are not evident immediately, but only become clear as we see them in the papers. The search string must therefore be adjusted each time a new keyword emerges as relevant or irrelevant. The main issue is therefore to discover a new important keyword when the review is very advanced. A late keyword discovery can result in many papers being added to the review and a subsequent delay in the publication of the results.

Table 11. Partial results of a term impact analysis on a search string

| **Full search string** | | |
|---|---|---|
| ("software development" OR "software engineering")<br>AND<br>("team programming" OR "team development" OR "pair programming" OR "pair development" OR "collaborative activities")<br>AND<br>($method OR $technique OR $pattern OR $style OR $variant) | | |
| **First concept validation** | **Results** | **Impact** |
| All keywords | 213 | - |
| Removed "software development" | 162 | Major |
| Removed "software engineering" | 121 | Major |
| No keywords (concept removed) | 621 | Major |
| Added "software process" OR "software processes" | 215 | Minor |
| **Second concept validation** | **Results** | **Impact** |
| All keywords | 213 | - |
| Removed "pair programming" | 57 | Major |
| No keywords (concept removed) | 83,594 | Major |
| Added "programming in pair" OR "programming in pairs" | 213 | None |
| **Third concept validation** | **Results** | **Impact** |
| All keywords | 213 | - |
| Removed $method | 118 | Major |
| Removed $variant | 213 | None |

Table 11. Partial results of a term impact analysis on a search string

| **Full search string** | | |
|---|---|---|
| ("software development" OR "software engineering")<br>AND<br>("team programming" OR "team development" OR "pair programming" OR "pair development" OR "collaborative activities")<br>AND<br>($method OR $technique OR $pattern OR $style OR $variant) | | |
| **First concept validation** | **Results** | **Impact** |
| No keywords (concept removed) | 563 | Major |
| Added $practice | 330 | Major |

### 3.4. Selection Process

Literature review processes often mix two quality aspects of the papers selected: quality in terms of the relevance of the paper for the purposes of the review, and quality in terms of the paper's underlying research methodology. The first quality aspect, which we call "relevance quality", is measured during the selection process. The second quality aspect, which we call "research quality", is described in section 3.5.

Relevance quality is determined by the selection process, specifically by the inclusion and exclusion criteria. These criteria define which papers are relevant and should be kept, and which papers are off topic and should be rejected.

#### 3.4.1. Selection process in the literature

It is apparent that the various steps of the selection process vary from author to author. Typical steps include:

1. Selection based on the paper’s title [81], [84];
2. Selection based on the paper’s keywords [84];
3. Selection based on the paper’s abstract [50], [81], [84].

On the inclusion and exclusion criteria, Brereton et al. [84] present a case where an exclusion criterion became evident only after the review was well under way. The aim of their review was to compare Pair Programming to Solo Programming, but they only realized during the review process that they should also exclude studies comparing different forms of Pair Programming. This shows that some of the more subtle criteria only become obvious once the domain is better known. Inclusion and exclusion criteria should therefore be periodically revised.

Brereton et al. also outline the difficulty of differentiating papers from studies [84]. One study can be presented in multiple papers, and, similarly, one paper can present multiple studies. This can cause a study to be overrepresented. Brereton et al. cite a specific study presented in five different papers. Inclusion and exclusion criteria must therefore define which of these papers should be kept and which should be rejected.

Another problem is the definition of overly restrictive inclusion and exclusion criteria, typical of a Google-style search. Rainer and Beecham explain it this way from their experience [77]:

*"Students tended to seek evidence to support the proposed intervention in their EBSE question, and tended not to seek evidence that contradicted the adoption of their proposed intervention or, alternatively, that supported the adoption of the baseline."*

However, based on another experience, Brereton et al. disagree with Rainer and Beecham on this point, reporting that, on the contrary [84]:

*"The student was very reluctant to exclude (irrelevant) studies which suggest that supervisors should pay particular attention to checking the results of this stage of the SLR process."*

It shows that novices can either be too restrictive or not restrictive enough in their choice of inclusion and exclusion criteria. Reviewers should therefore keep a close eye on the progress of the selection process, in order to ensure that the criteria are clearly and fully defined.

### 3.4.2. Lessons learned on the selection process

We propose the following two guidelines to help determine how restrictive the selection process should be at each step:

- The selection by title should not reject many papers, only duplicates and proceedings introductions. Very little information is given in a title, and therefore selection should not be about relevance at this stage.
- The selection by abstract stage is where most of the relevance quality should be evaluated.

We found that another step is needed for the selection process; a step we call the "overview". The overview step is based on the full papers, but limits the observation to the tables and figures, along with the conclusions of the paper. The main purpose of the overview step is to perform a final evaluation of the relevance along with a preliminary research quality evaluation (as

explained in section 3.5). This overview is useful when the abstract is poorly written, which is unfortunately often the case [50].

Clearly documenting the reasons for the inclusion or exclusion of papers is important when performing an SLR with a large group. Most review processes suggest the use of a spreadsheet application like Excel. Adding brief notes on the inclusion and exclusion of papers can help explain why papers were added or rejected, should questions arise later on. These notes can also help reviewers in the adjustment of their inclusion and exclusion criteria.

Inclusion and exclusion criteria must be crystal clear to all reviewers. If they are not, they cause large variations in the results and weak inter-rater agreement. Weak inter-rater agreement in the selection process must be discussed freely and openly with the reviewers, to establish the reasons for the poor results. These discussions enable the reviewers to synchronize their views on what constitute acceptable and unacceptable papers. In our experience, novice reviewers are rarely to blame for poor inter-rater agreement. The culprit is generally ambiguous criteria, where one or more criteria are interpreted differently by two or more reviewers.

Our two first reviews required that the selection be limited to empirical studies. The definition of what an empirical study is proved problematic, however. Can lessons learned be considered empirical? Does the post-mortem application of a new model on a previous study's data be considered empirical? These questions resulted in a number of ambiguities, because empirical studies were perceived differently by the reviewers.

We also learned that Excel is not entirely appropriate for collaborative work. Synchronization proved problematic, as the two reviewers worked in parallel. For a course-based literature review, every report deadline can be an opportunity to resynchronize the students' work with a "master" spreadsheet. We found many tools designed to help with systematic reviews, mostly in the medical domain. These tools could be used to better synchronize the work performed in software engineering reviews.

For our third review, there was a great deal of discussion on what constitutes a Pair Programming paper and what does not. Since the definition of the practice proved to be very flexible, we reconsidered many papers that had initially been deemed to be irrelevant. A careful study of the evidence showed that these borderline papers actually underscored a critical detail. The importance of this selection criterion did not emerge until the synthesis began. The need to structure the evidence showed that some evidence could not be categorized. These evidences that

cannot be categorized showed that a whole section of the literature has been neglected. A benefit of the iterative approach was to ensure that not all resources were expanded when synthesis began.

### 3.4.3. Problems to avoid during selection process tasks

The main problem is to exclude papers which might lessen the strength of the review's conclusions. As described in section 1.1., a systematic review should present an overview of the whole domain, and not just a subsection of it.

On the other hand, exclusion is also necessary to ensure a minimal relevance of the papers selected, since a large body or selected papers can become a burden during later tasks. Marginally relevant papers can provide some fragments of the answers to the research questions, but the work required to analyze and synthesize their information might not be worth it.

The selection process is therefore a compromise between the need to cover as much of the evidence available in the domain and the resources available.

## 3.5. Strength of the Evidence

This task is aimed at verifying the research quality of the selected papers. The conclusions of SLRs and EBSE reports are typically based on the highest quality papers, and use the lower quality ones only to add further support to those conclusions [41]. To achieve this, the research quality of the papers must be measured.

### 3.5.1. Strength of the evidence in the literature

Many authors [77], [84], [86] consider this task to be the most difficult to perform with novices. As Kitchenham writes [50]: *"Students found evaluating the quality of studies found in a systematic review particularly problematic."* As a consequence, Brereton et al. decided to skip this activity during their 2011 review [76]. As they describe it:

*"Students were not expected to assess study quality, an activity considered especially challenging even for experienced researchers."*

Their 2009 review blamed this problem on the fact that novice reviewers are not accustomed to the way empirical papers are written. Even the use of a simple checklist does not improve the situation. As Rainer et al. report [80]:

*"A number of students simply did not use the checklist, a number of students used it poorly, whilst some students used it well. The varied use of the checklist is surprising, as it had already been used in some tutorials and lectures."*

This means that the studies referred to above were evaluated based on the student's perception instead of on the evidence in the paper. The quality evaluation thus becomes a matter of personal opinion rather than a truly scientific endeavor.

### 3.5.2. Lessons learned on the strength of the evidence

A two-step approach is used for the research quality evaluations. The first step is the "overview" described previously in section 3.4. The research quality part of the overview consists in the verification of the presence of the most important context details. The context details include a description of the sample and its population, the study context, and clear study conclusions. A paper without these main details will certainly not pass the quality evaluation test, and will therefore be rejected.

The second step is a thorough quality evaluation, using a form based on the works of Dyba and Dingosyr [88]. Filling out this form requires a thorough reading of the full text, however, and this task is typically performed in parallel with the extraction (see section 3.6).
Mapping studies typically skip this step, as their goal is to provide a picture of all the studies performed in the field, and not only the good ones. We still recommend doing at least an overview of the selected papers, to ensure that the main axes of the mapping are based on solid evidence.

### 3.5.3. Problems to avoid during strength of the evidence tasks

The main problem is to base the conclusions of the review on dubious evidence. Editorials, theoretical models, exploratory research and the like have their place in the scientific process, but they should not make the bulk of the conclusions of a systematic review. The objective of a systematic review is to define what has been proved in a specific domain: Presenting dubious conclusions as proven facts can mislead the reader on the current state of the domain.

## 3.6. Analysis

In this task, the papers are analyzed in order to extract the relevant data. This analysis is typically performed using a standard extraction form, and sometimes aggregation tables, in order to facilitate the synthesis (described in section 3.7).

### 3.6.1. Analysis in the literature

Baldassarre et al. [81] reports that every paper was extracted by two students, who would then sit down together (with a domain expert) and build a single common extraction. A domain

expert was not available in our case so we could not validate it; however the practice of performing dual extraction and building a consensus is worthy of mention.

Petticrew and Roberts [[41], p165] recommend focusing on the studies of the highest research quality. The main synthesis work (and conclusions) should be based on these top quality papers, with the lower quality ones used for either confirmation or for minor conclusions. Analysis efforts should therefore focus on these high quality papers.

These authors [[41], p191] also recommend not transforming qualitative data into quantitative data. Practices like tally counting can be useful, but they result in a large amount of important context information being neglected, and should be used with care. In spite of the risk of losing context, tally counting and frequency analysis have been used in a number of studies [75], [84].

Brereton et al. [76] maintain that a "key problem" in data extraction is that students have to know which data are relevant, even though they are still domain novices. The extraction must therefore be adjusted as the need arises and as the students' comprehension of the field improves. Janzen and Ryoo [89] used a different extraction approach to avoid this problem: They asked each of their students to read 17 articles and summarize them. The synthesis was limited to producing a unified field summary out of the 17 paper summaries. The quality of the summaries produced proved to be equal to or better than that produced by experts.

**3.6.2. Lessons learned on analysis**

We found that our main problem was that the data extracted were not always what we needed for the synthesis. This is because we did not know beforehand which data we would require for the synthesis. In many cases, the extraction had to be redone once the synthesis approaches had been clarified.

We also found that providing novice reviewers with a completed form sample helped them understand what is required in each field. The form also needs to be sufficiently detailed in order to ensure that results from one reviewer to another are comparable. Reviewers should also focus on the conclusions of the studies, as they often provide the ‘meat’ of the synthesis.

According to qualitative feedback from the reviewers, this step is the most time-consuming activity of the process, but not the most mentally strenuous. In light of this, we recommend that extraction work start as soon as possible, in order to:

- Ensure that reviewers are familiar with the extraction procedures and the form, and what is required of them,
- Avoid a sudden drop in extraction quality toward the end of the review, when deadline pressures loom.

In addition, we found that some extractions were too succinct, and others too verbose. Constant evaluation and feedback on the results is therefore required in order to ensure that reviewers are all on the same page.

Another finding was that a paper selected by one reviewer can be deemed irrelevant by another reviewer. This indicates a conflict in the interpretation of inclusion and exclusion criteria. By justifying the reasons for inclusion and exclusion on the Excel spreadsheet, it is possible to understand what happened and make a correction if necessary.

We recommend that novice reviewers be assigned a short synthesis exercise based on good and bad extractions, ideally extractions performed by other reviewers. The goal of such an exercise is to raise the awareness of the reviewers as to what makes a good extraction and what makes a bad one.

For our third review on Pair Programming, the data extraction was performed in stages. During one iteration, a first batch of papers was analyzed and an attempt was made to synthesize the extracted data. The extraction form was modified based on the synthesis results. During the following iteration, the data extraction was corrected, and the analysis was then performed on a larger batch of papers.

This approach proved fruitful, as the final extraction form was very different from the initial one. Some data deemed important during the planning session proved to be of little value during synthesis, and a new synthesis approach had to be designed.

### 3.6.3. Problems to avoid during analysis tasks

The main problem is related to the amount of effort required for the analysis. We found that it is easy to spend all available resources to extract information of little use to the synthesis task. What seemed important during the initial stages can prove irrelevant when trying to structure the evidence. Rapid cycles of analysis and synthesis must therefore be performed until the extraction form stabilizes. When it becomes clear what we need to perform the synthesis, the analysis of the studies and the extraction of the information can be performed on a larger swath of papers.

### 3.7. Synthesis

For SLR and EBSE reports, the way in which the research questions are answered is described in this task. For scoping reviews, synthesis is limited to classification or mapping of the data.

#### 3.7.1. Synthesis in the literature

The activity of producing a synthesis has been reported as difficult by many authors [50], [81]. One of the difficulties stems from the use of data extracted by other reviewers. Baldassarre et al. describe filling out the data aggregation forms as *"a demanding task, as it requires combining the individual work of a number of students* [81]*."*

Felizardo et al. [90] pose an important question on the aggregation of data: Should the reviewers use tables or graphs? Biolchini et al. [10] favor the use of tables, while Kitchenham et al. [21] recommend graphs. The literature in the domain seems to indicate that some data are better represented in tables (e.g. tally counting), while other data are better represented in graphs (e.g. element comparison).

The serious synthesis problems described by Cruzes and Dyba [91] can be traced back to the proposed methods, which provide very few details on how to perform a synthesis. Current literature review processes do not describe how the synthesis could or should be performed. This is an important issue, since quantitative methods like meta-analysis can rarely be applied in a field like software engineering, which is rich in qualitative data. Consequently, there is a real need for a defined synthesis procedure for software engineering literature reviews.

#### 3.7.2. Lessons learned on synthesis

We found that students are typically expecting Google-like search responses, which are responses that provide the expected answer to the research questions. As a result, they discard some articles as irrelevant, even though they provide interesting answer fragments.

The synthesis approach we use and which provided the best results was one of successive summarizations. This approach follows these steps:

1. Based on its extraction form, a selected paper is summarized in a single paragraph.
2. This paragraph is further summarized into a single phrase.
3. The phrases from all the selected papers are put together to see how they support or contradict each others.
4. A single paragraph is built based on all the selected papers' phrases.

At each stage of the approach, cohesiveness of the result must be monitored. The phrases and paragraph built must make a coherent whole.

The final paragraph produce by this successive summarization exercise does not consider the context of the study, and should not be used as-is in the synthesis of the review. The objective of the exercise is limited to finding a potential structure to the evidence found in the selected papers. We found that this simple exercise helped many students when determining the structure of the evidence.

This exercise also found relevant differences in the capabilities of the students: Some students were able to produce a very coherent paragraph, while others could not go beyond a mere word-for-word copy of the main finding of each of the papers. It is not clear why the difference between the good synthesis and the bad ones is so large: A poor understanding of the domain and the research question can certainly be blamed, but we think that writing a good synthesis requires an inherent talent for seeing the big picture, a talent not possessed by everyone. This hypothesis will need to be confirmed with a more formal experiment.

Another synthesis method we used is the Grounded Theory approach [74], which can help reviewers categorize the evidence. Describing this approach is beyond the scope of this paper, however we can comment on a few main points:

- The categorization can result in categories which are too generic and contain nearly all the papers. Similarly, some categories can be too specific and contain only one or two papers.
- A large amount of coded evidence without a category might indicate that a category is missing.
- The upper level of a categorization scheme must have a suitable number of elements. Too many upper-level concepts make the scheme difficult to understand, while too few make it too generic to be useful. We found that 5 to 7 concepts at the highest level proved the most useful, which is in line with Miller's recommendation [92].
- Providing a categorization example to the students can be a mistake, since a major bias toward this example may result.
- Providing an upper-level model for the categorization scheme before the codification work has been completed may also cause a major bias to appear in the results.

Feedback from the students underlines the fact that the synthesis activity is not especially time-consuming, but is very mentally strenuous. The successive summarization approach helps the students in finding the relation between the evidence, but still imposes a significant mental strain.

During the third review, we also learned that synthesis is improved with the use of iterations. It took many synthesis attempts on partial data extractions before the reviewers obtained a good perception of the domain, and so were able to synthesize it accurately.

### 3.7.3. Problems to avoid during synthesis tasks

The main problem of the synthesis is to present conclusions which are not representative of the domain. Biases can be introduced at every step of the systematic review, but synthesis is especially vulnerable to it because of its basis in human judgment.

Some systematic review authors mitigate that problem by using a validated model to structure the evidence. For example, Bjornson and Dingsoyr structured their evidence on knowledge management on the recognized taxonomy of Earl [93]. This enables the summarization of the evidence on a supposedly exhaustive and non-biased framework.

## 3.8. Process Monitoring

This task is related to the collection of data during the systematic review in order to improve the process for future reviews.

### 3.8.1. Process monitoring in the literature

Only the EBSE process mentions the need for process improvement [21]. The goal of EBSE being the selection of a technology and dissemination of the technology within a specific context, the process improvement task is focused on gathering feedback on the use of the newly introduced technology. Its goal is therefore centered on ensuring that the review conclusions are properly used.

For SLR in general, the process improvement task should cover a wider context. Feedback on the review conclusions should be considered, but feedback should also be gathered on the reviewers themselves. The discovery and diagnostic of issue should not be limited to the review's conclusions, but should also cover any details available on the specific tasks of the review.

### 3.8.2. Lessons learned on process monitoring

This paper is the result of multiple improvements on the available systematic review processes. These improvements were made possible through the gathering of data at each and every task of the process. Reviewers were required to provide the time spent on the tasks, and to comment on the difficulties faced. These reports enabled us to find problems and propose solutions to common issues of systematic review processes.

Solutions like the term impact analysis for search string validation, and the successive summarization approach for the synthesis of the evidence were introduced because of reported issues with these tasks.

### 3.8.3. Problems to avoid during process monitoring tasks

The main problem is to fail to learn from previous errors. Systematic reviews are complex processes which can fail at any stage: Either fail to provide quality results, or fail to perform the work within the budgeted resources.

## 4. Discussion

In this section, we present two remarks from the literature and from our own experience which are not related to a specific task of the review process.

### 4.1. On Effort

Literature reviews, whether they are SLRs, scoping reviews, or EBSE, require a great deal of effort. There is near-consensus in the literature that reviewers have limited time available to conduct SLRs [50], [75], [77], [78]. As Rainer and Beecham point out [75]:

*"You do not have the time, effort and resources available to undertake a full-scale evaluation using Evidence Based Software Engineering. Therefore you need to think carefully about the time, effort and resources you do have available and how those resources should be allocated to the different steps of your EBSE evaluation."*

However, it is not clear which activities require the most effort or why. The amount of effort required for an activity can depend on many factors: the domain structure, the number of sources available, the complexity of the questions, the background of the reviewers, their motivation, etc. Informal evaluations showed that the Analysis activity proved the most time consuming, but the Synthesis proved the most mentally strenuous. We therefore recommend that each student fill out a short form and attach it to their deliverables, reporting on the hours worked, the strain required, and the problems faced. An evaluation in the style of NASA-TLX

[94] could be an interesting way to measure strain. These data would provide information on the process, which can then be adjusted accordingly.

### 4.1. Threats to Validity

The main threat to the validity of the iSR approach concerns the revision of the research questions and of the selection procedure during execution. It is very easy for a team of reviewers to introduce a confirmation bias or a publication bias by writing favorable research questions and inclusion/exclusion criteria. Reviewers should keep a careful eye on these two aspects of bias, in order to ensure that the conclusions still represent the state of the whole domain, and not the state of some unrepresentative subsample of papers.

## 5. Conclusions

Lessons learned from the systematic reviews demonstrated that an iterative approach can be beneficial when working with domain novices. Literature reviews require a large amount of knowledge, a challenge similar to that of many software development projects. Reviewers should therefore follow the recommendations of software development process experts and use an iterative method instead of a gate-based waterfall method. As the review advances, the perception of the domain by novices change, and the design of the review should evolve accordingly. The research questions may have to be rewritten, the selection procedure may need some adjustment, the extraction forms and analysis tables may require revision, and the synthesis conclusions may need to be redesigned. This approach should produce better and more accurate results iteration after iteration, reflecting the progressive gain in expertise on the part of the novices. It should enable reviewers to calibrate the effort output required through the addition or removal of iterations. In addition, the iSR approach should make it possible to improve a review that ended with only partial results, through the addition of more iteration.

## Acknowledgment

We thank Raphael Aujame for devising the "term impact analysis" method that we used for search string validation. This research was supported in part by NSERC grant A-0141.