FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Investigating programming techniques in large scale empirical studies

Experiences and lessons learned from coding contests

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Martin Kitzler
Matrikelnummer 0525135

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: ao.Univ.-Prof. Dr. Stefan Biffl
Mitwirkung: Dipl.-Ing. Dietmar Winkler

Wien, 02.09.2014

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Investigating programming techniques in large scale empirical studies

Experiences and lessons learned from coding contests

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering/Internet Computing

by

Martin Kitzler
Registration Number 0525135

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: ao.Univ.-Prof. Dr. Stefan Biffl
Assistance: Dipl.-Ing. Dietmar Winkler

Vienna, 02.09.2014

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Martin Kitzler
Am Schulberg 42, 2124 Niederkreuzstetten

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

A special thanks I want to give to my supervisors Stefan Biffl and Dietmar Winkler, who provided me with a vast amount of support.

Another thanks goes to Catalysts GmbH, in particular Dr. Christoph Steindl, who hosted the coding contest that was used in the experiment, for the uncomplicated and efficient collaboration.

Beside the fact that I wrote this thesis alone, thanks to the support, I am using the plural personal pronoun “we” for the rest of the thesis.

Abstract

Empirical studies and experiments provide the foundation for empirical research in academia and industry. Typically, planning and executing controlled experiments require a lot of effort on preparation and execution.

This thesis introduces the concept of using public coding contests as a vehicle for scientific studies. Coding contests are typically large events where skilled developers are trying to solve problems as fast as possible. By setting up rules that participants have to decide and choose particular SW development techniques (of a small predefined selection) to solve a problem, allows empirical analysis afterwards.

The benefits are: researchers obtain data from a large set of subjects for their studies, while saving effort by not having to host the experiment themselves. A contest organizer can profit from it, by getting free advertisement and by polishing the reputation of their contest with a scientific touch.

To test our idea, four research issues are stated and discussed in the later sections. The first two discuss the vehicle concept in general, under a scientific point of view. For the last two issues we conduct a proof of concept by embedding an experiment into a real coding contest with hundred of participants and analyze the results, namely, the effects of Pair-Programming (PP) and coding experience on time and defect variables.
Pair-Programming is a software development techniques that is an actual topic under investigation in the field of computer science. However, studies about the efficiency of PP have varying results and still need further replication.

The methodological approach for this thesis was to start with a literature research about experimentation in SW engineering and Pair-Programming, followed by the conduction of an experiment. The results of the experiment are used to discuss the research issues.

As result, the experiment was performed successfully and delivered enough data to discuss the research issues. Pairs needed the same time to finish their tasks than individual programmers which means that the effort (person hours) doubled. In regard to software quality (defects), pairs competed slightly better but not significantly.

The target audiences are a) researchers in the field of empirical software engineering and b) organizations using comprehensive recruiting methods or assessments to test their future employees.

Kurzfassung

Empirische Studien und Experimente bilden das Fundament für empirische Forschung in der Wissenschaft und Wirtschaft. Typischerweise bedeutet die Planung und Ausführung kontrollierter Experimente einen hohen zeitlichen Aufwand.

Diese Diplomarbeit stellt das Konzept vor, Programmierwettbewerbe als Vehikel für wissenschaftliche Forschung zu verwenden. Programmierwettbewerbe sind üblicherweise große Veranstaltungen wo Entwickler versuchen Aufgaben so schnell wie möglich zu lösen. Ein Festlegen in den Regeln, dass die Teilnehmer sich für eine (von mehreren vorgegebenen) Entwicklungsmethodologien entscheiden müssen, ermöglicht empirische Untersuchungen im Nachhinein.

Vorteile für den Wissenschaftler sind, dass er/sie, auf Grund der hohen Teilnehmerzahl, mehr Daten für seine Studien bekommt, während er/sie von dem Aufwand das Experiment selbst abzuhalten entbunden werden. Ein Organisator eines Programmierwettbewerbs profitiert durch freie Werbung und einer Aufwertung des Bewerbs, da es einen wissenschaftlicheren Anstrich erhält.

Um unsere Idee zu testen werden in den späteren Kapiteln werden vier wissenschaftliche Fragen untersucht. In den ersten Beiden, wird das Vehikelkonzept von einem wissenschaftlichen Standpunkt aus untersucht. Für die zweite Frage wird ein Machbarkeitsbeweis durchgeführt: Es wird ein Experiment in einen realen Wettbewerb mit hunderten Teilnehmern eingebettet und die Auswirkungen von Paarprogrammierung und Erfahrung auf Zeit- und Fehlervariablen untersucht.

Die Paarprogrammierung ist eine bekannte Software-Entwicklungstechnik die derzeit noch im Zentrum wissenschaftlicher Untersuchungen steht. Die Studien über ihre Effizienz führten zu widersprüchlichen Ergebnissen und bedürfen weiterer Replikation.

Der methodische Ansatz dieser Arbeit erforderte eine Literaturrecherche über empirische Wissenschaft im Bereich der Software-Entwicklung sowie der Paarprogrammierung. Im Anschluss wurde das Experiment geplant und durchgeführt. Das Ergebnis des Experiments bot die Grundlage zur Adressierung der Forschungsfragen.

Das Experiment wurde erfolgreich durchgeführt und lieferte genügend Daten zur Behandlung der Forschungsfragen. Paare benötigten gleich lange wie Einzelprogrammierer weshalb zur Folge der Zeitaufwand (Personenstunden) verdoppelt wurde. Bezüglich Softwarequalität (Fehler) schnitten die Paare geringfügig besser ab, allerdings ist das Ergebnis nicht signifikant.

Das Zielpublikum ist a) Wissenschaftler im Bereich der empirischen Software-Entwicklung und b) Organisationen die umfangreiche Rekrutiermethoden oder Assessment-Center verwenden um zukünftige Mitarbeiter auszutesten.

Contents

CHAPTER 1

Introduction

This chaper will start with a motivation about the topic of this master thesis.

Later in this chapter we will provide the necessary information to understand the context. In the second section, software development methodologies will be defined. We start with a broader focus and then specialize down to the topic of Pair-Programming.

The third part "Empirical studies in Software Engineering" explains how empirical science in software engineering works. We introduce papers and books that provide the base on how studies should be planned, performed and published.

Coding contests form the last topic of this chapter. We present how they are set up and list up established contests.

1.1 Motivation

The idea of this master thesis was using coding contests as a vehicle for empirical studies. The high number of participants in coding contests will provide many data points that can be analyzed and interpreted. Additionally, researchers are relieved from most organizational tasks if the contest is hosted by another company or institution.

We wanted to analyze what is necessary to conduct a scientific study in this context and thus worked out four research issues, split into two topics. The first issues focuses on how the experiment can be set up and the second one how the experiment can be used to draw scientific conclusions from it. For the later two issues we wanted to use actual research topics in the field of computer science that are under investigation. The choice fell on Pair-Programming and the effects of experience on software quality. Pair-Programming is a SW development technique that is used by various SW development methodologies.

In the later part of this chapter we will provide the necessary information base to understand the context of the research issues and the topics under investigation.

1.2 Software Development Methodologies

According to Pezze and Young [51] and Dyck and Majchrzak [21] "Software development methodologies (SDM) provide frameworks for managing development projects. They do not primarily address technical details of how software is developed but determine the organizational embedding of development and give guidelines for organizing activities." In other words, SDMs are frameworks that help organizing and executing the software development process of a project to decrease the chances of failure.

Dyck and Majchrzak [21] further summarized, that the first development methodology was the 'Phase model' in 1956 by Benington and since then many others have been made up.

E.g. Well known SDMs are the 'waterfall model', the 'Rational Unified Process' by IBM (1998), Scrum (since 1995) and 'Extreme programming' by Beck in 1999 [8]. 'Structured programming' and 'Object-oriented programming' are programming paradigms but can also be seen as early SDM approaches.

Starting with the waterfall model, software development evolved over the last decades, passing incremental and spiral approaches and currently reached 'Agile software development' which was first mentioned in 2001 in the 'Agile Manifesto'[1]. The manifesto reads as follows:
"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more." For a more detailed explanation see the following link [2].

Summarized: Agile software development is a subsection of SDMs which use iterative and incremental workflows. The key meanings behind 'agile' are, that requirements change during the development process but the teams handle these changes by rapid responses. Secondly, the communication inside the team and the customer is very important: Short daily meetings shall eliminate problems inside the team quickly and there is frequent communication with the customer representative to receive regular feedback to newly finished features.

Popular agile SDM approaches are Scrum and Extreme programming. Also often used in agile SDM are techniques such as Pair-Programming and test-driven development. This approaches and techniques are further described in the following subsections.

See also [20] for a review about development of agile methods in the last decade. Also note that these SDMs are still topics in active research. [30]

[1] http://agilemanifesto.org/
[2] http://www.ambysoft.com/essays/agileManifesto.html

1.2.1 Extreme programming

Extreme programming (XP) is a popular agile process which was developed in 1996 by Kent Beck, Ward Cunningham and Ron Jeffries and published, in 1999, in their book 'Extreme Programming Explained: Embrace Change' [8].

XP focuses on customer satisfaction, this can be achieved by small development cycles and prioritizing on the important tasks first. The developers are encouraged to deliver software that the customer really needs instead of stubbornly sticking to road maps and other documentation. In other words, the team should focus on the important, urgent needs and think if they really need a new feature before implementing it. This is called the YAGNI (You ain't gonna need it) approach.

Coding and testing are valued above exaggerative planing. Prototypes should be developed early and cover the crucial parts of a project. Further, automated unit tests should be written while implementing new features. If sticking tightly to the test-driven-development practice then the test case should be written before the actual code fragment. Testing also includes code reviews, meaning one developer presents his/her code to a small group of other developers to show them how the new features are solved and also to receive feedback and criticisms. This also helps the developer to improve his/her skills.

Flexibility is valued highly in XP, meaning that developers should not stick to already written code too much. When the requirements and design change then the code should get refactored to the new needs without hesitation or delay (even late in the project).

XP also suggests that teams should split up into pairs and perform Pair-Programming. This should spread project knowledge and improve the code quality.

See also [65] for more detailed information about XP.

1.2.2 Scrum

The development of Scrum began in the 1990's when the agile SDM ideas were rising up. According to Gloger [25] Scrum overtook XP in popularity and became de-facto standard for agile software development projects, because it has no technical specifications (like XP) but a more general, versatile approach.

Scrum is agile because it uses an iterative and incremental approach. Unlike traditional software development, the process is cut into so called 'sprints' which cover a timespan of multiple weeks. The goal is to deliver working software to the customer at the end of each sprint. Sprint planning meetings are held before the start of sprint to define realistic goals.

Every morning of a sprint interval the team gathers for around 15 minutes where each team member summarizes 1) which tasks he/she finished yesterday, 2) which tasks he/she is planning to accomplish today and 3) if there are road blockers on the way and help (from other team members) is needed. These daily meetings are called 'daily scrum'.

At the end of each sprint cycle the team gathers and holds review- and retrospective-meetings. Review meetings focus on the project and tasks, e.g. tickets that could not be completed, while retrospective meetings are there to discuss the development process itself, e.g. where improvements are needed.

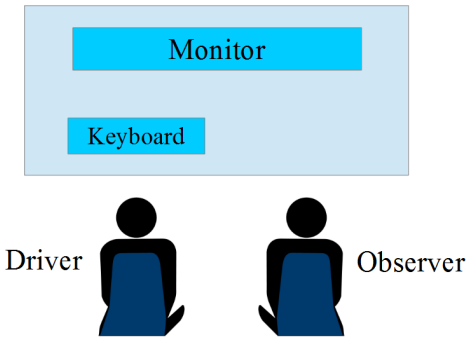


Figure 1.1: A simple diagram showing a typical Pair-Programming workstation. One person acts as driver and controls the keyboard and mouse while the other person acts as observer (sometimes called: navigator) to indicate mistakes of the driver. Note that the monitor is between both actors. The source for the clipart was openclipart.org.

Each Scrum project should have a person being the 'Scrum master'. This role is slightly from the typical project leader. The Scrum master enforces, that the Scrum process is being applied correctly (e.g. daily meetings) and tries to resolve other influences that could disturb the development team and prevent them from achieving the sprints' goal.

Like XP, Scrum expects that requirements and specifications of a project change and thereby encourages the team to adopt to these changes quickly. Scrum further tries to enhance the teamwork by letting the team work together on a common goal. When a cycle is over and all sprint goals are accomplished the developers can cheer up because they accomplished something as a team.

1.3 Software Development Techniques

In the last chapter we discussed SW development methodologies that provide general, non-technical solution on how to build software. Software development techniques provide more specialized methods that can be used inside SDMs. We will describe Pair-Programming, as the focus of our research issues will lie on it, and test-driven-development, which was also used in our experiment.

1.3.1 Pair-Programming

Cockburn and Williams(2000) [15] describe PP as "Two programmers working collaboratively on the same algorithm, design or programming task, sitting side by side at one computer." Williams et al. [61] further explanation: One person acts as the 'driver', he/she sits in front of the computer, controls the mouse and keyboard and writes code. The other person, the observer, sits next to the driver and supervises the drivers actions. I.e. the observer does not produce code him-/herself but only checks the code of the driver and speaks up when seeing a mistake. Ideally

the driver vocally comments what he/she is doing so that the observer can easier see his/her intentions. Before starting both coders can also discuss how they plan on solving their upcoming tasks and agree on a theoretical solution. Driver and observer should switch roles in regular time intervals (e.g. two hours). It is also recommended to change the pairing partner regularly. [66]

The aim of PP is primary to improve the code quality. When working together, mistakes are probably identified earlier or prevented at all.

According to Williams et al. [61] Pair-Programming was first observed by Larry Constantine in 1995 [16]. Also in 1995, Jim Coplien presented the 'developing-in-pairs' organizational pattern. Pair-Programming is a fundamental part of Extreme Programming (See previous chapter). However, it can be used in any software development process. [59]

For several decades, PP has been described as a practice that improves the quality of a software product, shortens development time, increases the motivation and productivity of programmers and spreads knowledge between team members. [61] [47] [60] [34] Many empirical studies and experiments have been made to evaluate PP's advantages and disadvantages, summarized by two meta studies of Hannay et. al in 2009 [29] and Salleh et. al in 2011 [56]. However, the meta studies [29] [56] did not support all of PP's stated advantages. They report that some experiments had a low quality and thus their results are questionable. Generally most of the studies had a small sample size (i.e. lower than 30) and were held with students only. The quality of the produced source code is hard to measure and depends on the evaluating experts. Studies about the number of defects produced by pairs vs. single programmers lead to contradictory results, stating that it depends on the experience, motivation and background of the pairing programmers.

They agree that PP reduces the development time but the factor of how much it is different per study is in most cases below 50 % which means that the overall effort (i.e. person hours) is increased. Experiments with students came to the predominant conclusion that PP increases the learning effect and exam scores of students. E.g. see Williams et al. [62].

1.3.2 Test driven development

Test driven development (TDD) is a development technique introduced by Kent Beck in his book 'Test driven development: by example' [9] in 2003. The main principle of TDD is to write test cases (TC) before implementing new features. By doing so, the programmer focuses on writing code that makes the new TC pass. Thus, TDD tries to improve the productivity of the developer.

Beck's [9] TDD phases in detail are:

1. Write a test case. The developer writes a test cases that cover the new feature or defect that he/she tries to implement or solve. Complex TCs should be split into multiple smaller ones so that problems can easier be identified when a TC fails.

2. Run all test cases. After writing the TC the complete test suite is run and the new test case is expected to fail. Running a TC that tests not implemented features are still useful to check if the feature may have been implemented already by somebody else, as well as checking that the test definitely fails, because if the new test would pass then it is likely that the TC is incorrect.

3. Implement. After the TC is written (and failing) the developer starts implementing the new feature. The aim is to only write code that is needed for the TC to pass. Thus, the developer doesn't waste time by producing code that is not necessary.

4. Rerun all test cases. Once the feature is implemented, the test suite is run again. Running all tests will show if the new code affected existing features. When there are tests failing then the next step is to analyze why they fail and fix them (go back to step three).

5. Refactor. When the new feature is implemented and all tests are passing then the existing code will be refactored to fulfill particular coding standards. In other words: step five only polishes the code produced in step three. After step five is finished the developer will continue with a new task in step one again.

1.4 Empirical studies in Sofware Engineering

To evaluate SW development techniques, empirical research is needed. In this section we provide basic information on how empirical studies are performed and which methods are commonly used.

Wohlin et at. [64] describe SW Engineering as following "Software engineering is a cross-disciplinary subject. It stretches from technical issues such as databases and operating systems, through language issues, for example, syntax and semantics, to social issues and psychology. Software development is human-intensive; we are, at least today, unable to manufacture new software." They further state, that even though a development process is human-intensive, based on the creativity and ingenuity of the programmer, we should still treat it as science and thereby use scientific methods when doing research about the ways of how to create software.

Juristo and Moreno [33] highlighted the impact of the human factor on software development "Another important constraint on running experiments in SE is the effect of the human factor on software development; that is, SE is not a discipline whose result is independent of practitioners. So, the result of several people applying one and the same software artefact (technique, process, tool, etc.) will almost certainly yield different results. This amounts to a substantial obstacle to generalising the results yielded by empirical testing."

Basili [5] presented and Glass [24] summarized four research methods for the field of software engineering (SE):

- The scientific method: Observe the real world and build a (simulation) model.

- The engineering method: Study current solutions and evaluate changes.

- The empirical method: Propose a model, develop measurement methods and apply these to case studies or experiments.

- The analytical method: Propose a formal theory, derive results and if possible do empirical observations.

The empirical method was not given much attention in the field of SE in their starting decades. Basili, followed by others, started in the 1980's to stress for the need of experimentation. [64] Claims for the need of experimentation continue further until the 2000's like in the publication of Wohlin et al.'s book [64].

1.4.1 Empirical methods

There are three empirical methods, described by Robson [55]:

- Case Study. A case study is usually a planned study which is running in parallel with a real project. Unlike experiments, the researchers have no control on the execution of the project and thus only collect data.
- Survey. Based on literature or some knowledge base a survey can be performed using questionnaires or interviews.
- Experiment. Experiments are made to test (previously defined) hypotheses using statistical tools. Unlike case studies, experiments are run in controlled environment where the researcher has full control over the execution.

According to Wohlin et al. [64] an experiment is structured into five phases:

1. Definition. The definition is made to clearly formulate the objective goals of an experiment. If focuses on the object of study, the purpose, the quality focus, the perspective and the context.
2. Planning. The planning lays the foundation for the experiment and defines the context in detail. I.e. Choose the environment for the experiment (e.g. university, industry) and the subjects needed; Formulate the null and alternative hypotheses; Choose the dependent and independent variables, define their measurement scales and constraints for later statistical analysis; Fixate the experiment design and instrumentation; Consider threats to validity.
3. Operation. The operation phase is divided into a) Preparation, i.e. prepare subjects and material for the execution; ensure that the participants are committed for the experiment. b) Execution, i.e. actual experiment; Enforce the rules of the experiment. c) Data validation, i.e. Ensure that all data is collected correctly.
4. Analysis and interpretation. In this phase the data collected in step three is analyzed using descriptive statistics. Secondly, consider removing data points or reduce the number of variables. As the next step perform the hypothesis tests and finally, interpret if the hypothesis will be rejected or accepted.
5. Presentation and package. The last phase of an experiment is to present or package the results, e.g. via publishing research papers. This phase is important because a proper presentation and documentation allows replications of the experiment.

1.5 Coding contests

For our empirical analysis we selected coding contests as a vehicle for our experiment.

Coding contests are events where software engineers are trying to solve logical or mathematical problems according to a specification in a competitive environment. Thus, it is sometimes referenced as 'competitive programming'.
The environment and rules of the particular events (see list below in section 1.5.1) vary, but the aim of the contests is always one or a combination of the following points:

- solve the problem(s) as fast as possible
- solve as many problems as possible
- deliver as efficient as possible solution(s) (E.g. shortest path)
- maximize the programs quality (E.g. failure-tolerance, lines of code)

There can also be restrictions on the participation-team-sizes, age of the participant, number of workstations, programming languages, allowed software tools and libraries, membership of certain universities, association or countries, etc.

Motivations for the participants can be trophy money, fame (and prestigious job offerings) or the hunt for a thrill.

The competitions usually take less than a day and can be online-only or bound to physical locations where each contestant has to attend. Large contests consist of (online) qualification rounds where the number of participants are reduced and a grand final where the best qualifiers fight for the title.

1.5.1 List of established coding contests

ACM's International Collegiate Programming Contest (ICPC) [3] The ICPC is one of the oldest contests and exists in its current form since 1977. In 2012, 25,000 students from over 2,200 universities attended at the regional qualification rounds and the top 112 teams competed at the world finals. [54] This contest is a student only competition. Students of same universities have to form teams of three and find a coach at their institution. Some universities have a local contest first, to identify the most competitive teams which will be sent to the regional contests. The best teams of the regional contests will be sent to the finals.

Each team only has one workstation to work on so that the schedule of which program is typed when has to be well thought through. Each task has to be solved completely to achieve points.

International Olympiad in Informatics (IOI) [4] The IOI is a contest for young students or pupils only, since the maximum age for participation is 19. Unlike ICPC, each country (not university) can send up to four competitors to the annual contest which is held at a different

[3] icpc.baylor.edu
[4] www.ioinformatics.org

place each year. There each contestant has to work on his/her own and solve the given tasks. In contrast to ICPC also partially correct submissions are awarded marks. [18]
During the two five-hour sessions, no ranking is published. Afterwards, the best 1/12 coders will receive a gold medal, the next 1/6 a silver medal, followed by the next 1/4 to receive bronze.

Google CodeJam [5] This competition from Google contains of four rounds where individual programmers solve algorithmic problems. To attend the first round a qualification has to be completed which takes around two hours. Phase one has still an unlimited maximum of competitors, but one has to be in the top 3000 in the 2.5-hours phase to proceed to phase two. Round two and three each last another 2.5-hours and the top 500 respectively 25 advance to the next or last round. The final takes another four hours.

TopCoder [6] "TopCoder5 is a crowd-sourcing contest where developers (of no restriction to institution) can submit solutions for a defined problem. The results are used to identify the best overall solution, which can be used by companies and/or governmental organizations." [63] The problems of TopCoder are similar to IOI or ICPC problems but two of the tasks are typically less algorithmically. [18]

Unlike other contests, the competition is executed online. Each programmer has access to an online system where his/her source code is written, compiled and tested. However, it is possible to copy the code and work on the local machine and reinsert the code for submission. The contest is split into three phases: coding, challenge and system test. The coding phase lasts 75 minutes, afterwards the challenge phase begins where the competitors can view the code of other programmers and construct test cases to challenge their program and receive extra points if one can bring another program to fail. Phase three is the system test round; the programs that survive all system tests are scored.

Challange24 [7] Challenge24 is a contest like ICPC for teams of three, without any restrictions to age of institution. After a qualification round which is held online, the best 30 teams are invited to Budapest to compete in the finals. The final is a 24-hours contest where 6-8 tasks have to be programmed. The Ch24 is well known for their innovative tasks. [63].

Catalysts Coding Contest (CCC) [8] The Catalysts Coding Contest by the Austrian company Catalysts GmbH allows participants to choose whether they prefer working on their own or in teams. All participants work on a set of consecutive tasks with increasing complexity within a level-based structure. This level-based structure (the levels have to be solved in sequence) enables a direct comparison of the performance of all participants/teams per level. In addition there are no restrictions that hinder participating.

Since 2007 the contest grew continuously up to 300 participants on site. For a detailed description of the CCC see section 5.1.

[5] www.google.com/codejam
[6] www.topcoder.com
[7] ch24.org
[8] www.catalysts.cc/contest

1.6 Summary

The listed software development methodologies, especially Pair-Programming, still need further research and study replication to falsify or not falsify its claimed effects. By using the represented empirical methods, we will use a coding contest, namely the CCC, as a vehicle to run a controlled experiment.

CHAPTER 2

Related Work

In this chapter we will describe the state-of-the-art in the fields of Pair-Programming and coding contests. The chapter is split into sections that are grouped by the various aspects of Pair-Programming and coding contests. In each section a selection of studies and a summary are given. The focus of this chapter lies on the differences in duration/effort/defects between pair programmers and individual programmers as well as studies that are built around coding contests.

The aim of this chapter is to give the reader a summary of the related work and provide the base for the research issues in the following chapter.

2.1 Experimentation in Software Engineering

In this section we will review experiments, in particular empirical-experiments, in the field of Software Engineering (SE).

Basili [6] stated in 1985 that software methodologies and technologies need to be evaluated by metrics. He discribes various models that can be used to show the advantages and weaknesses of methodologies. Thus, the methodologies or tools can be selected for a project based on the metrics. Secondly, the goodness of it can also be measured during the process where it is used.
Basili, et al. [7] also discussed the need for "families of experiments" to build up knowledge in software engineering. Their suggested procedure is to organize empirical work into the form of families of related experiments. Thus, 1) the effects of alternative variable values in the model can be investigated, 2) the strategy of how detailed hypotheses are investigated can be varied and 3) certain threats to validity that often arise can be dealt with. A second conclusion of them is that, empirical research has to be a collaborative activity because the huge number of problems, variables and issues to consider.
The work of Basili et al. will be used as orientation on how our experiment shall be planned and documented.

In 2000, Wohlin et al. [64] published a book where the process to conduct experiments is explained in details. The book already achieved several editions and is the starting point for many

researchers when planning to conducting empirical experiments. This master thesis' experiment is also based on their book. (See chapter 5)
Their book explains all the necessary steps that are needed to ensure scientific standards. (See chapter 1)

Jedlitschka and Nick [32] emphasized the need of researchers to share their experiment results with each other. Much information lies bunkered in departments of companies or universities that should be shared.
They describe the interest of companies to install knowledge repositories. In detail they review established tools in the industry, e.g. ESERNET, and how their working model fits into experience based information systems.

Empirical ecosystems are inspired from software-ecosystems (SECO) which are interactions of businesses and services related to a common technology. Manikas and Hansen [37] give a systematic literature review on this topic by analyzing 90 papers in detail. Based on these papers they provided an overview of the definitions of SECOs and the common main items that consist a SECO.
Example of a SECO: Google's Android operating system. Google provides the basic software, independent developers are providing apps and normal users use Android and apps. Further there are other services like software support, stores, etc. The different actors are interacting with each other, based around a software or platform.
They observe, that this differs from traditional outsourcing techniques because the initiating actor neither hires the actors nor owns the software they produce.

Biffl et al. presented a paper in 2013 [11] with different approaches of replication data management (RDM). RDM describes ways of how to unite data collections of different experiments or experiment iterations to use them for comparison. Their suggested way was to use empirical ecosystems, i.e. researchers use common data models for the experiments and share their local repositories.
The term "empirical ecosystems" is created in relation to "software ecosystem", meaning that like software that is interacted with in SW ecosystems, experiments should be interacted with by various researchers in empirical ecosytems. When a researcher publishes the results of an experiment, other researchers can use the same data and verify it. The verification should be seen as an equal important step as the first publication.

Jedlitschka and Nick [32], Manikas and Hansen [37] and Biffl et al. [11] are used for an orientation on the current state of the art about empirical ecosystems. Our work will orientate on the suggestions made in the cited papers.

2.2 Empirical Studies on Pair-Programming

In this section we present studies about Pair-Programming, grouped into the topics: duration & cost, quality improvements, effects of experience and transferring knowledge.

2.2.1 Costs and Duration of Pair-Programming

In this subsection, studies about the costs and duration of PP are presented, followed by a summary which highlights research gaps.

Studies Nosek [47] was one of the first to publish the results of an empirical experiment about PP in a scientific journal. In his experiment, 15 professional programmers from the industry were randomly grouped into five pairs and five single programmers. They all came from the same company and were very familiar with the IDE. Pairs and individuals had 45 minutes to solve a task in a controlled environment.
In the experiment the following variables were looked at: readability, functionality, duration, confidence and enjoyment.
As results, Nosek reports that pairs significantly completed more functionality, pairs had significantly more confidence of their work, pairs significantly enjoyed the work more. Unsignificant was the finding that pairs were 43 % to complete their work and that the code of pairs was more readable.
This paper is often referred to, however, the sample size is very small (15), the experiment only lasted 45 min which means the problem could not have been very complex. There is no accurate data about the experience of the participants and their team history. It can be expected that they knew each other and maybe programmed in the same teams already. An explanation for the very positive result towards PP could be the stated fact that no "pair jelling" was required, the small experiment size and random distribution of the participants may led to an expert-expert-pair vs. junior-individual situation.

Williams, Kessler, Cunningham and Jeffries [61] conducted a university experiment with students to confirm Nosek's [47] results. 28 students formed 14 pairs, while 13 students acted in a control group and worked individually at theirs workstations. The two groups were created, based on the students grades to provide an equal distribution. During the experiment (course) they had to write four programs which were written in controlled environments. Most of the participants had a lot of coding experience as well as interest in PP.
They observed the following variables in their experiment: defects, duration, confidence and satisfaction.
Significant results were that pairs completed the tasks 40-50% faster, pairs produced less defects, pairs had more confidence of their work and pairs enjoyed their work more.
Opinion: This paper is often referred to. The sample size of Williams's (et al.) experiment is bigger than Nosek's [47] and supports his statements about PP. Although, most of the participants were undergraduate students and 85% of them favored Pair-Programming which might effected the results because the pairs had a motivation boost. University experiments with students make it hard to draw conclusions to the general population. This paper also discussed the effect of pair jelling: On the first task, pairs were 20% faster than individuals and on the second and third task this advance increased up to 40-50%.

Cockburn and Williams [15] summarize the research about PP so far and focus on the economic advantages. They refer to other papers stating how expensive it is to deliver fixes for defects later. Based on Williams et al. (2000) [61] the total effort (person hours) of projects using PP is only about 15% more than projects using individual programmers. The number of

defects decreased by 15% as well.
PP also increases the satisfaction of the programmers, citing one student of the experiment: “It’s nice to celebrate with somebody when something works.”
Regarding coding quality they showed a diagram representing the lines of code (LOC) of the experiment [61] pointing out, that the program’s LOC of PP-teams is about 20% less than of programs from individual programmers. They stated further that combined with the reduced LOC the programs had a higher quality as well.
The numbers mentioned are based on Williams et al. (2000) [61] student experiment and cannot be adopted to an industry setting 1:1. However, they quote professional programmers, who are questioned about their PP experience in industry projects, saying that they profited from PP. This paper is a very positive review of PP pointing out all the advantages with no or declining disadvantages due to the overwhelmingly positive effects.

In 2001, Nawrocki and Wojciechowski [46] made a university experiment consisting of 21 students of the same experience level. The students were divided into three groups: 1) The participants in the first group worked individually by applying PSP (Personel Software Process). 2) In the second group a variation of XP (Extreme Programming) was used where no PP is applied. 3) Participants of the third group used XP with PP. The tasks were not complex and needed 150 to 400 LOC (C/C++) to solve. Four programs had to be written in a controlled environment and the results were evaluated. The participants program is checked when they submit it, and if it is not producing the correct results, they have to resubmit it again. This resubmission-counter is used as an indicator for defects.
For their experiments they took a look at the following variables: duration, LOC, defects.
The results (not described if significant or not significant): PP is slighty faster than individual programming but not the 50% mentioned by Williams et al. [61]. The number of resubmissions (defects) is slighty smaller in PP-teams. Single programmers produced more LOC per hour.
This study contrasts to the other studies because in the experiment, PP did not achieve the excellent results mentioned by Williams et al. [61] and Nosek [47]. However, the sample size is again quite small (21) and consisted of students only. A meaningful link to industry situations cannot be drawn. Furthermore, the program sizes were small (150-400 LOC, 2-4 hours per program) which may reduced the positive effects of PP. Its unclear if the participants knew each other before, but with the short duration of the experiment there may have been no “pair jelling” which should drastically increase the performance of teams. [60] The number of defects produced by PP were only slightly smaller, combined with the only slightly faster development time, PP is a very expensive process that has only small advantages compared to single programming.
An interesting fact mentioned in the paper is, that PP had the lowest standard deviation on the average variable values. Nawrocki and Wojciechowski [46] summarize that this makes PP the most predictable process (of the three processes used in the experiment) regarding to development time.

Padberg and Müller [48] analyzed when it is useful to use PP in industry projects, focusing on metrics of three categories: 1) process metrics (like the pair speed advantage), 2) product metrics (as module breakdown structure) and 3) project context metrics (such as market pressure). They state that the personnel cost is basically doubled when working in pairs but on the other hand pairs work faster and produce less defects.

They came to following conclusions:

- "If the market pressure is strong, the faster time to market of Pair-Programming can balance the increased personnel cost"
- "One important task for future research about Pair-Programming is to collect reliable empirical data about how large the pair speed and defect advantage actually are; currently, empirical evidence is very limited"

Opinion: They summarize PP and ask the important question "Is it worth it?". However they quoted they need more reliable empirical data about the speed and defect advantage which means their results are not applicable in a general context.

Parrish, Smith, D.Hale and J.Hale [49] used a new approach towards PP: Based on earlier studies they state that pair programmers are more productive than individuals. To research why PP is more productive, they reuse data from one of their earlier experiments. 48 modules that have been developed by professional developers in two-man-teams were selected and their productivity factor (average number of unadjusted functions points completed by time unit) calculated. Note that the teams did not use PP, instead they worked on two different machines. The developers stated how much they worked concurrently or not concurrently on the same module. Using this concurrency factor they split the modules into two groups: high and low concurrency. The group with low concurrency was four times more productive than the group with high concurrency (statistically significant). Parrish et al. state, that since PP works even though there is a high concurrency, is does not work in teams using no PP while having high concurrency. This means that the benefits of PP must arise from its role-based-protocol (driver, navigator).
"We conclude that the role-based protocol prescribed in Pair-Programming overcomes a natural productivity loss from working in pairs."
In their experiment they used data of programming pairs that worked on two separate machines. They did not compare their productivity factor with actual PP data. With their experiment they wanted to test if this productivity gain comes from concurrency only or not. Their results indicate that teams become less productive if the concurrency is high, PP however overcomes this productivity loss.

Phongpaibul and Boehm [53] made two experiments to evaluate the effects of PP. In the first experiment 70 undergraduate students participated during a software engineering class. They were assigned to 14 five-man-teams based on their grades. In total each team needed 400-800 man-hours to complete their task.
In the industrial experiment, eight developers were divided into two teams which both had to program a similar web application. Both teams spent 1340 - 1400 man-hours in total.
Results from the student experiment: PP teams needed significantly less man-hours to complete their program (24% less), there were no significant differences between the number of un-passed test cases or incomplete requirements between the PP and solo programming team.
Results from the industry experiment: the PP team needed 4% less man-hours than the single programming team, the PP team made significantly less major defects (39% less).
This experiment shows the so far best results for Pair-Programming because according to Phongpaibul and Boehm [53] PP takes even less effort than single programming (Respectively 23%

and 4%). This is a huge difference to other experiments that claimed that PP would need about 15% more effort. [15] However, the number of participants in the students experiment is high enough to be meaningful, while the results from the industry projects cannot be taken granted due to their small number. On the other side the man-hours used were very high compared to other experiments, increasing the plausibility of their findings.
Their excellent results still seem unrealistic. The authors [53] stated that the results could have cultural reasons: "The experiment was conducted in Thailand where the culture is much different from the country in America or Europe. Thai people socialize more than western people do. Pair development required continuous interaction between the developers, which may not be a practical approach in US where people generally require more personal space."

Arisholm, Gallis, Dybå and Sjøberg [3] conducted an experiment with 295 professional Java consultants from 29 companies in Norway, Sweden and the UK. In 2001, the first part of the experiment was held: 99 Java consultants were asked to perform several change tasks on Java systems with different degrees of complexity. In the second part of the experiment, 2004-2005, Arisholm et al. [3] repeated the first part with 196 other subjects (again Java consultants) but they grouped them into 98 pairs. For both parts they used the same experimental procedure and material. The pairs have been constituted, based on their experience (junior, intermediate or senior) forming equally skilled pairs: juniors with juniors, etc.
They analyzed the following variables: system complexity, experience, duration/effort and correctness
The results indicated that the effects of PP depend on a combination of programmer expertise and system complexity.
Junior pairs needed 4-6% more time (109-112% more effort) to complete the task but achieved a much higher correctness than individuals: 32% for easy tasks and 149% for complex tasks. The authors suggest that if a junior has to do complex maintenance tasks then they should perform it in pairs.
Intermediates were 16-39% quicker than individuals (22-68% more effort). For simple tasks the correctness was 29% lower but for complex tasks 92% higher than the correctness of programs from individuals.
Seniors needed 23% less for simple tasks and 8% more time for complex tasks (55% and 115% more effort). The correctness decreased by 13% (simple tasks) and 2% (complex tasks) when compared to individuals.
Opinion: The experiments sample size is very large with 295 participants which shows the effort that had been made by Arisholm et al. The way the two parts of the experiment were conducted seems very well planned and the results have been interpreted honestly and realistically. This is also the largest experiment with professionals that had the aim to investigate the effects of PP.
However, there are some factors that limit the meaningfulness of their results:

- The experiment parts lasted only a day which means that the programming tasks could not have been very complex. Other papers state, that PP really pays off once the pairs are 'jelled', which means the partners came to knew each other and understand their way of thinking. [61] [35]

- Another pair jelling problem could be the cultural background. The subjects came from

different countries with different mother tongues. Although they may all spoke English perfectly, it can still limit the productivity if you work with a complete stranger from another country without some time to get to know each other. This has been mentioned by authors themselves as well.

- The programming environment may had been completely different than the developers are used to. It is not written which IDE or other tools had to be used. Maybe the pairs had different IDE experiences which reduced they development speed. What about other things like keyboards? Norwegian, Swedish and British alphabets are different from each other - as well are the keyboard layouts. Considering the short experiment duration it is dubious that the subjects could have used their full coding potential during the experiment.

- Between the two experiment parts there was a timespan of three to four years. In 2001 the individuals solved the tasks and 2004/05 the pairs. This can mean changes in the developer environment (higher versions) which might inflicted inconsistencies the results.

Bipp, Lepper and Schmedding [12] performed an experiment with 70 students during software development university classes. The 70 students were split into nine groups of 7-8 group members. Five of nine groups were asked to use PP while the other four groups had no restrictions. Before the experiment, the students programming experience was investigated and no significant mean differences was found between the subjects. One PP group was removed during the experiment because they refused to work together.
For their experiment, they analyzed the following variables: LOC and quality.
Results: PP teams produced as much LOC as paired teams in the same amount of time, the code quality of PP teams was higher: 1) LCOM (Lack of Cohesion in Methods) was lower but not significantly lower in PP teams. 2) RFC (Response for a Class) was lower, but not significant lower for PP teams. 3) WMC (Weighted Methods per Class) was lower in three of four PP groups but not significantly.
This paper is interesting because they focus on the quality of program code produced by PP-teams and individuals-teams. On average the LOC of the programs was 3500-4000 (without GUI) once it was finished, which shows that they were complex enough to be meaningful. The result of the LCOM comparison clearly states that pairs produce superior code quality. Regarding RFC and WMC the pairs had been better in three of four cases but the authors [12] mention that the one individuals team, that achieved better results, performed generally very well. Bipp et al. also state, that RFC and WMC show obvious differences between quality of PP vs. individual programming but the sample size was too small to be significant.

Lui, Chan and Nosek [36] criticized older experiments that program design related tasks have not been taken into account to evaluate the performance of PP. Two quotes to express their motivation:

- “One cannot just measure how long it takes for pairs and individuals to write the same program as language skills and tool usage skills can significantly interfere with performance and mask defects in program design.”

- "It is important to know what we are actually measuring: Is it the productivity of problem solving and designing algorithms or the knowledge of computer languages and IDE familiarity?"

To measure productivity gains in design tasks Lui et al. [36] chose IBM-PAT tests. PAT stands for Programmer Aptitude Test and they are designed to measure programming aptitude without testing knowledge of specific languages. These tests focus on procedural problems, classification knowledge, deduction questions and mathematical reasoning. In their experiments, the authors use PATs to disentangle design related tasks from coding to isolate the performance of pairs and solos. They used the REAP variable to compare the effort needed between pairs and solos to complete their tasks (See page 25 for an explanation of REAP).
In the first experiment, 15 part-time undergraduate students that worked as full-time industrial programmers had to solve procedural-solution-tasks. Ten of them in five pairs, and five as solos (randomly assigned). The participants had to (re)submit their procedural solution until it was correct. The authors measured the resubmission-count and the duration needed to complete the task.
Results of the first experiment indicate that based on the average time to complete a task: REAP is -1.1%, but based on the accumulated submission times REAP was -1.8%. Pairs needed 2.8 resubmissions and individuals 3.4 to successfully complete the task.
The second experiment consisted of ten of the previous 15 students, forming five pairs. Iteratively, each pair had to solve a deductive problem as a pair and after they found the solution both pair-members had to solve a similar problem independently. Once both solved the independent tasks, the iteration started again with a problem to solve as a pair, et cetera. Lui et al. measured [36] how many times the pairs and solos had to resubmit their solution until it was correct and how many minutes it took them.
The results of the second experiment show, that REAP was 22.9% based on the average elapsed time, or -5.3% based on the accumulated submission time. According to the average time needed, the second experiment's tasks were more complex than in the first experiment: Pairs in Exp1 needed 16.2 minutes and in Exp2 83.3 minutes, individuals in Exp1 needed 33.6 minutes and in Exp2 135.5 minutes. Pairs needed 1.8 resubmissions and individuals 3.3 on average to successfully complete a task.
Opinion: Lui et al. [36] illuminated another aspect of PP, namely: What are we testing if we let a programming pair compete against a solo in coding? Do we test the impact of PP or the developers IDE knowledge? This paper follows the promising idea to test pairs and solos in their program design skills instead of coding skills. The results are encouraging: REAP was -1.1% in the first experiment and 22.9% in the second. In both experiments the number of resubmissions needed to complete the tasks were lower by 18% and 45%.
The paper shows that PP outperformed single programming regarding design tasks. However, the sample size in both experiments were small and secondly, the PAT approach cannot prove that PP is generally more efficient than solo programming by only looking at the design part of a project.

Hannay, Dybå, Arisholm and Sjøberg [29] performed a meta analysis on 18 empirical experiments about the effects of Pair-Programming that published their results about quality (defects) and/or duration and effort differences. All of the papers taken into the study had to provide

enough information to standardize the results to allow a comparison of the results.
Of the 18 experiments, eleven had been made in Europe, seven in North America and consisted of twelve to 295 subjects participating in it. 13 studies were conducted with students only, four with professionals only and one with a mix of both.
Hannay et al. focused on 1) quality, i.e. number of test cases passed (defects), 2) duration to complete the task and 3) effort, i.e. the double value of the duration to compare the man-hours.
In their study they presented a forest plot that shows the results of the studies used for their meta analysis. The forest plot shows, that there is a small positive overall effect of PP compared to solo programming. Only one of 14 studies showed a negative effect on quality. The overall fixed model shows a value of 0.23 and the 0.33 in the overall random model.
There is a medium positive overall effect of PP on duration. (Overall fixed model: 0.40, overall random model: 0.53) Two of eleven studied showed a negative effect on duration.
The overall effect of PP on effort is medium negative. (Overall fixed model: -0.73, overall random model: -0.52) Seven of nine studies showed a negative effect on effort. The two studies showing a positive effect were both written by the same authors.
The authors also state, that there are signs of publication bias. By using Duval and Tweedie's "trim and fill" method they draw funnel plots and analyze if the plots are symmetric or how the plot would look like if it would be symmetric. They review, that for quality and duration there are studies missing that report of negative effects of PP for quality and duration.
Hannay et al. [29] also review the different results achieved by PP regarding experience and complexity.
Results: "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": It depends on other factors, for example, the expertise of the programmers and on the complexity of the system and tasks to be solved." [29]
Other results are:

- Junior pairs performances were close to that of senior individuals.
- Juniors might increase their performance by using more verbalization. They cite other studies that concluded that the positive effects of PP come from the increased amount of verbalization instead of the role driven scheme (driver, navigator).
- PP is beneficial when trying to achieve a higher correctness of highly complex tasks.
- "Pair-Programming may also have a time gain on simpler tasks." [29]
- "The higher quality for complex tasks comes at a price of a considerably higher effort (cost), while the reduced completion time for the simpler tasks comes at a price of a noticeably lower quality." [29]

For future work, the authors suggest to analyze group dynamics and pair dialogs to understand what makes pairs more and less efficient.
Opinion: This paper was created with great effort and combines most of the empirical researches

until 2009 and combined their results. Hannay et al. [29] did a great job to bring together the relevant studies and analyzed them regarding quality, duration and effort effects of PP. However, the results are disillusioning since the effects did not achieve the high benefits that were predicted by other papers [47] [61]. This might have team collaboration reasons: Did the pairs work together well? Was there enough time to get to know each other to understand how the other one works?

Salleh, Mendes and Grundy [56] performed a systematic literature review on empirical studies about the effectiveness of PP in 2011. They focused on teaching effects of PP in higher education like exam grades and student motivation but they also reviewed performance and quality aspects. In their review, they accepted 73 studies for further investigation.
Three factor groups were analyzed in detail:

- **Compability Factors** 14 factors about compatibility were identified in 17 studies. The most investigated factor was 'personality' with two studies observing a significant positive effect, five studies no significant effect and two studies with mixed findings.
  The second factor was 'actual skill level', i.e. programming experience, academic background and academic performance, where nine studies saw a significant positive effect, two a significant negative effect and one no significant effect.
  The third factor 'percieved skill level' signals the skill of the subject relative to that of the skill of his/her partner in the pair (subjective). Four studies found a significant positive effect for PP in this factor. For both skill levels the studies had an consensus that PP works best when both students of a pair have a similar skill level.
  The other factors were 'self-esteem', 'gender', 'ethnicity', 'learning style', 'work ethic', 'time management ability', ' feelgood factor', 'confidence level', 'type of role', 'type of task' and 'communication skills'. These factors had only one or two studies describing them with either significant positive effects or no significant effects. Regarding confidence, one study reported that very talented programmers tend to work alone and enjoy PP less, however another study contradicted these results. One study suggests that the task type significantly affects PP's effectiveness. See Salleh et al. [56] for the other factors' details.

- **Measure of Effectiveness** Factors for effectiveness (actually efficiency) were grouped into another four categories: Technical productivity, program/design quality, academic performance and satisfaction.
  31 studies measured technical productivity. Of this 31, 19 evaluated 'time spent' as a indicator for PP's efficiency, eleven studies of these 19 report that PP is faster, seven complain that PP needs more effort due to more man-hours.
  31 of 45 studies that compared PP to solo programing, report an improvement in technical productivity and satisfaction.
  8 of 16 studies report a significant positive effect of PP on academic performance.

- **Measure of Quality** 32 studies investigated in this area. The most important categories are: Internal and External code quality. The results about internal code quality showed mostly no effects or mixed ones. External code quality was reviewed by 16 studies, nine of them found a significant positive effect, five no effect and two mixed ones.

Opinion: Although this study focuses on using PP for educational reasons, most of the studied aspects are valid in general, with the only constraint that all subjects were students. The results regarding quality and performance indicate that PP achieves better results. Unfortunately this study didn't review the 'effort' of PP compared to single programming. That PP is faster was already shown by Hanney et al. [29] but the effort is very important since it represents the 'costs' of PP.

Summary Table 2.1 shows studies evaluating the effect of PP on duration and effort, compared to single programming. The mean result on duration was a 40.67% faster development process and the mean effort needed increased by 18.67%. The meta study from Hannay et al. [29] shows a positive effect on development time as well, but didn't state concrete values. On effort Hannay et al. summarize a negative effect. Salleh et al. [56] report in their meta study of 19 studies that saw a decreased development time (lower duration) when using PP programming and 7 studies that complain that PP still needs more man-hours (higher effort).

The meta studies give evidence that PP increases the development speed of programmers. The speed increase factor depends on many factors like how long the pair members worked together (pair jelling [61]), their experience with programming, the complexity of the task, personalities, etc., but based on the study results so far the factor is at around 40%. This means that pairs are 40% faster but still need 20% more total effort in person hours, to complete the job.
Studies suggest [15] that combined with the lower defect rate of pairs, PP is cheaper. Other studies [48] deny that and state that PP is still more expensive. Only if the time-to-market pressure is high, then PP is a very useful tool to gain a competition advantage against other companies developing a similar product.
However, most experiments focused on students only and especially the earlier experiments had very small sample sizes. E.g. Phongpaibul06 [53] industrial experiment consisted of eight developers. Thus, there is still need for further replication with subjects from industry and with higher participant numbers.

Arisholm et al. [3] suggest that future experiments should include larger, more complex tasks and widely mixed pairs with different skill and experience combinations.
Many studies relying on students as subjects suggest for future works that more experiments with professionals in industrial settings should be made. [42] [61] Most of the experiments were also made with a small subject size.

2.2.2 Quality Improvements with Pair-Programming

In this subsection, studies about quality improvements (defects) by using PP are presented, followed by a summary which highlights research gaps.

Studies Müller [42] conducted two experiments to evaluate if PP or solo programming combined with peer-code-reviews achieve the better results. In 2002, 20 students (5 pairs and 10 solos) completed two programs that need 200-300 LOC to complete during computer science classes. In 2003, the experiment was repeated with another 18 students (4 pairs and 10 solos). The pairs were grouped by experience so that each pair and solo had an average programming

Study	Duration	Effort
Nosek98 [47]	43% faster	14% higher
Williams00 [61]	45% faster	10% higher
Nawrocki01 [46]	23% faster	54% higher
Müller05 [42]	43.5% faster	13% higher
Phongpaibul06 [53]	62% faster (students) 52% faster (professionals)	24% lower 4% lower
Arisholm07 [3]	8% faster	84% higher
Lui08 [36]	50.5% faster (1st exp) 39% faster (2nd exp)	1% lower 22% higher
Mean	40.67% faster	18.67% higher

Table 2.1: Summary of studies evaluating the effect of PP on duration and effort.

experience of 6-7 years.
He analyzed the following variables: defects and effort.
As results, he stated that pairs and solos (with peer reviews) produce a high and similar level of correctness, if forced to (significant). A not significant result was, that pairs need 13% more person hour effort to complete the implementation.
Opinion: The sample size with 20 respectively 18 students is too small to draw meaningful conclusions that impacts PP in industry. The two experiments differ slightly in the results: In the first experiment the costs for quality assurance is higher for the solo programming groups, while in the second experiment it is cheaper (both not significant). Furthermore the program sizes were small with 200-300 LOC and the students working in pairs most likely didn't know each other before which means that pair jelling wasn't given at the first and maybe second program. Another point is, that there have been two different lecturers in both experiments teaching the students (One for the PP groups and the other one for the solos) which may lead to internal threads to validity as Müller states himself. [42]

Hulkko and Abrahamsson [31] summarized the empirical evidences for PP so far (1998-2004) and criticize, that most of the experiments were made in a too small context and only with students as subjects. Therefore they made a case study about four real SW development projects where the impact of PP was studied. The projects took between five to eight weeks and consisted of four to six programmers. In the end the LOC of the four different projects varied from 3700 to 7700.
In their experiment they used the following variables: productivity, defects and code quality.
As results, they claimed that productivity varied between the projects leading to no indication that PP has a higher one. Further, there are conflicting results about the defect density between the projects. Regarding quality: pairs have a lower adherence to coding standards but pairs have a higher comment ratio.
Only three of the four mentioned projects had metrics about the productivity between pairs and solos. They measured productivity in logical LOC per person hour. In two of three projects PP had the better ratio but they interpreted it as a conflicting result.

Defect ratio was only measured in two of the four projects. In one of them PP and solo programming had an equal amount of defects per thousand LOC (KLOC), while in the other project PP had six times fewer defects per KLOC than solo programming. Again this was interpreted as a conflicting result.
Coding standard adherence was documented in two projects. The result was that PP had 1.5 respectively 2 times more deviations from the projects coding standards. This was interpreted as a contradiction to the supposed advantages of PP.
Comment ratio was measured in three projects. In one of them the ratio was equal between PP and solo programming but in the other two projects PP had an around 33% higher comment ratio.
Although they were declared as industry projects, two of them consisted of students only while the other two were a mix between students and professionals. There is nothing mentioned on which criteria the pairs or teams were created and how the programming experience was spread. The projects took at least five weeks, so pair jelling should not have been a problem but the big question is why the defect density differed so much in the two projects that measured it. In the project were the defect density was equal between PP and single programming there is a suspicious decrease of the usage of PP inside the project: Starting from around 95% in the first to iterations it dropped to 20% in the fifth iteration. There is no information in the paper why the team decided to switch to single programming.

Di Bella, Fronza, Phaphoom, Sillitti, Succi and Vlasenko [19] [52] analyzed the work of 17 industrial developers for 14 months to provide evidence of the effects of PP in industrial settings. In their case study they found out, that the defects tend to decrease when PP is used. This effect, that PP decreases the introduction of new defects is perceivable but not significant.
Opinion: Unfortunately this quite new case study didn't find out results that differ much from previous works. PP only seems to provide a small bonus regarding code quality (defects).

Summary A summary of the presented studies about the affects of PP on defect density can be found in table 2.2. Six of ten papers state that the teams using PP had fewer defects in their programs (7-48%). There is no exact number for the Nawrocki01 study [46] but PP had a positive effect in their experiment regarding numbers of resubmissions needed to pass all tests.
Two studies reported an equal amount of defects in the code, stating that PP did not have an impact on code quality. Müller05 [42] concludes that PP and single programming (SP) have equally high quality if SP is combined with code rewiews that force the coders to produce higher quality.
Hulkko's (2005) [31] experiment came to conflicting results regarding defect density. In one of the two groups in his experiment PP had six times fewer defects than the solo team members but in the other group the defect rate was equal.
The last study in the table, Bipp08 [12], did not analyze the amounts of defects but the design quality of the source codes. They found significant differences between single programmers and pair programmers in LCOM (Lack of Cohesion in Methods), RFC (Response for a Class) and WMC (Weighted Methos per Class) all benefiting PP.
The average percentage of defect rate improvement was 18.14%.

Six of nine studies about defect density showed an improvement of defects density in the

Study	Effect on quality
Cockburn01 [15]	15% fewer
Nawrocki01 [46]	slightly fewer
Müller05 [42]	equal to SP with reviews
Hulkko05 [31]	conflicting results
Phongpaibul06 [53]	equal to SP (students) 39% fewer (professionals)
Arisholm07 [3]	7% fewer
Lui08 [36]	18% fewer (1st exp) 48% fewer (2nd exp)
Bipp08 [12]	positive on LCOM positive on RFC positive on WMC
Mean	18.14% fewer

Table 2.2: Summary of studies evaluating the effect of PP on defect density.

code with an average improvement rate of 18.14%. Two studies reported neither improvement nor worsening and one came to no results since it was contradicting. The reason why it was contradicting is probably due to the small sample size and few groups. When looking at this results it seems clear that PP has a positive effect on defect density. However, how much better is still unclear: The effect varied between seven and 48%.
Hannay et al.'s meta study [29] also report of a positive effect of PP on quality. 13 of 14 evaluated studies had a positive effect. They conclude, that PP benefits the most for highly complex tasks but did not state estimation how much it will bring.
In Salleh et al.'s meta study [56] four studies reported a positive effect of PP on defects but two studies showed no effect. Two other studies showed a decrease of defects when using Pair-Programming, on the other hand, two papers had conflicting results.
The review about the effects of PP on defect density shows an improvement of 18.14%. This high number relies mainly on Lui et al.'s [36] experiment with 48% improvement, but was conducted using undergraduate students as subjects only. Secondly on Phongpaibul et al.'s experiment [53] which shows an 39% improvement using professionals, but the participant count for this experiment amounts to eight persons only. Thus, 18.14% seems too high for a general assumption.
There is a need for further large scale replication of testing PP on students and professionals.

No exact number can be told on how much PP decreases defect density. This still relies on various factors like expertise, motivation, pair jelling, etc. but the study results vary a lot and two studies reported on no effects at all. Future experiments should make more pair member variations [3] and contain more study subjects, since the sample size in most of the experiments was small.
Other studies like Xu and Rajlich [67] presented case studies which have to be evaluated in a quantitative way.

2.2.3 Impact of Experience on Pair-Programming

In this subsection the presented studies relating to the impact of experience and programmer-background on PP will be discussed.

Studies Lui and Chan [35] compared the three experiments of Nosek98 [47], Williams00 [61] and Nawrocki01 [46] using the variable REAP (Relative Effort Afforded by Pairs). REAP describes the relative effort of man-hours needed between programs of pairs compared to programs of individuals. A REAP value of 0% means the man-hours are equal, while 100% means that PP needs double the man-hours of single programming. In the Nosek98 [47] experiment REAP was 42%, in the Williams00 [61] experiment 15% and in the Nawrocki01 [46] experiment 100%. Lui and Chan [35] criticize that the students in the experiments were grouped into pairs by their academic records and not by programming experience.
They conducted an own experiment using 24 part-time masters students who had full-time programming jobs. They were grouped into eight pairs and eight solo programmers. The aim behind the experiment was to compare the REAP between novice-novice-pairs and novice-solos and between expert-expert-pairs and expert-solos.
Their idea was: if a developer has to write a program where the problem is new to him, it makes him a novice regarding to experience. If the developer has to write the same program again from scratch, he would be better and faster. Once he did it even more often he would be an 'expert' for this problem.
In Lui's and Chan's [35] experiment the participating teams and solos had to code a program about the same problem multiple times. After the first stage, REAP was 29% (Pairs needed 410 minutes, solos 635 minutes). Each round both groups improved their results but the solos caught up with the pairs on the fourth round. I.e. REAP increased to 57% in the second round, 69% in the third and 91% (pairs 272 minutes, solos 285 minutes) in the fourth round.
Lui's and Chan's [35] conclusion is, that PP is worth the effort if the problem is complex for the developers experience (novices). If the problem is known to the developers (experts) than PP does not pay off.
Quotes: "As long as a pair knows a programming solution well enough, it is effective for the person who controls the keyboard and mouse not to interrupt his writing even though he may make small mistakes such as typos. On the other hand, his partner probably feels less challenged by watching the known solution the guy is writing."
"We conclude that novice–novice pairs against novice solos are much more productive in terms of elapsed time and software quality than expert–expert pairs against expert solos."
Opinion: The approach that doing the same things multiple times in a row makes a developer equal to an expert seems a stretch. It is obvious that the navigator of a pair will get bored if he watches the driver writing the same code the fourth time. Furthermore, at some point both groups (pairs and solos) will be approximated to each other, because the task stopped being a challenge but started to be only a 'paperwork'.
In summary, an interesting approach but it only predicates that when tasks are repetitive then single programming has the lower effort.

Summary Three of the studies above used professionals as subjects (Nosek98 [47], Phongpaibul06 [53] and Lui08 [36]) and regarding effort the results were 14% higher, 4% lower and 11% higher. The other studies' experiments were made with students only (mostly undergraduate).
Lui and Chan [35] made a study where they compare novice pairs against novice solos and expert pairs against expert solos with the result that novice pairs bring a higher productivity boost than expert pairs.
Hannay et al.'s meta study [29] presented a table, where the effects of PP are grouped by the programmers expertise. Their findings are, that junior pairs have the highest effort (increase by 111%) but also the highest correctness increase (73%). Especially in complex systems (see graph (f) the correctness for junior pairs increases by 149%. Intermediate pairs had the lowest effort increase (43%) and achieved only a low increase in correctness: 4% in total, -29% for simple tasks, 92% for complex tasks. Senior pairs had an increase of effort in 83% and the correctness decreased by 8% (-13% for simple, -2% for complex tasks).
Salleh et al.'s meta study [56] analyzed the effects of the actual-skill-level and perceived-skill-level on participants on PP. Nine of twelve studies reporting about actual skill level say that it significantly improved PP's effectiveness (Two had significant negative results and the last study found no significant results). Salleh et al. findings about perceived skill level suggest that PP works best when both pairs partners have a similar skill level. They also grouped the results of the effects on personality on the effectiveness of PP with the result that most studies (five) found no significant effect, two studies found positive effects (pairs are more productive if the two partners have mixed personalities and temperaments) and two studies found mixed effects.

The results from the studies presented using developers as subjects are slightly better than the overall results for student experiments, but this can be due to the small sample sizes and the low number of studies.
Lui and Chans [35] findings fit with Hannay et al.s meta study [29] that when pairing novices the level of productivity or quality increase is higher than when pairing experts.
According to Salleh et al.'s meta study [56] the best results should be achieved by pairs that have the same experience level but different personalities.
Brought10 [14] did not present results but state that if coders with a different level of experience are paired together, than the stronger one takes the lead and the weaker student does not necessarily learn from him.

Like already mentioned in the future-works-sections above the number of experiments made with professionals from industry is low and the existing experiments have a small sample size (except for Arisholm et al. [3]). However the Arisholm experiment neglected the effects of pair jelling to some degree since the pair members did not know each other before.

2.2.4 Knowledge Transfer and Learning

This subsection summarizes the effects of PP on other areas like: teaching and learning, coder motivation and distributed Pair-Programming. These studies are reviewed to understand further impacts that PP can have on programmers and might affect outcomes of experiments if neglected.

Teaching and Learning McDowell, Werner, Bullock and Fernald [40] conducted an experiment with 485 students of the University of California - Santa Cruz. In an introductory programming course 141 students wrote their programs independently and 344 students worked in 172 pairs. The teams were assigned by the lecturers based on the students preferences. In the fall 2000 semester the PP part of the exerpiment was held, while the individuals were tested in spring 2001. During the course(s) they had to complete five assignments that were comparable in complexity.
Experiment variables were: programming assignments score (functionality + readability) and the final exam scores.
Significant results of the experiment were, that pair's programming score is better by ~20% and that pair members have a higher chance to take the final exam.
A not significant result was individuals having a higher score at the final exam.
Opinion: The high number of participants make the results of their experiment convincing. Functionality and readability of pairs scored 86.3% while individuals scored 67%. McDowell et al. [40] explanations are: "First, it may be that pair-programming enhanced the quality of the output resulting in programs that were more functional and readable. A second possibility is that the mean programming score in the pairing class was artificially inflated. Because both members of the pairs earned the same grade on each of the programming assignments, overall scores in the class may have simply reflected the performance of the stronger student in each pair". To disprove the second possibility they compared the programming score of the PP teams (86.3%) with the score of the better 50% of the individuals (77%), showing that still, the PP score is significantly higher than individuals.
Individuals received higher scores on the final exam (not significant), but according to McDowell et al. this could be explained by the lesser attendance of individuals in the final exam (significant, pairs 92.4% vs. individuals 75.5%). Pairs seem to have a higher confidence of their skills, or a higher motivation due to the 'group pressure' to attend at the final exam.
However, there are factors that could limit the conclusiveness of the results: The two parts of the experiment were held in different semesters which may affect the participants performance. The programming assignments may not have been equal. Furthermore the pairs did not work in a controlled environment.
In a second paper McDowell et al. [41] suggested to use PP to improve the confidence of programmers which may also be used to increase the interest of women for programming.

Williams, McDowell, Nagappan, Fernald and Werner [62] published results from their experiments in various papers. [61] [40] In this work they merged the results and reported their findings about PP. There were eight experiment samples of the North Carolina State University (NCSU) and four at the University of California - Santa Cruz (UCSC). In total 1200 participating students. The focus of the study laid on using PP as a learning technique.
As variables in their experiment, they used: course success rate, final exam score, project score and attitude.
They claimed a significant benefit for PP in course success rate; a not significant difference in the final exam scores between PP and individuals; diverse results about project score: no significance at NCSU but significant benefit for PP at UCSC; and a significant positive attitude towards PP.

Opinion: The results between the samples of the two universities differ at the final exam score and project score variables. In two samples of the NSCU project score there is no exact number which teams used PP. The authors also state that "It is possible that the NCSU differences are not significant due to the unstructured pairing arrangements". Like mentioned above, the paper also focused on PP as a learning technique using mostly junior programmers as participants.
See also [23], [58], [45] and [44].

Braught, MacCormick and Wahls [14] analyzed data from 259 students of a computers science class to compare their performance. During four years, 13 classes had been held by four different instructors. In total, 142 students were paired by their ability, 41 students were paired by random and 76 students worked individually. They measured 1) weekly written homework assignments, 2) weekly programming assignments, 3) three written exams and 4) two programming exams. Braught et al. [14] conclude that PP may improve the performance of less able students that have similar abilities.
Quotes: "One might be tempted to pair a weaker student with a stronger student so that the weaker student can learn from the stronger. But in our experience, and in that of others [4, 10], the stronger student instead takes over, and either does the work or simply gives direction."
Opinion: The research is based on a previous study of Braught, Eby and Wahls. In this previous study they found out that PP seems to let the weakest students benefit the most. When reviewing the results they focus on the lowest quartile regarding scores. The upper three quartiles didn't show significant differences and have been mostly left out of the study. However this is a very restricted approach and the title is irritating.

Summary Williams00 [61] reports of a significant positive benefit due PP on the course success rate. Hanks06 [26] mentions that the results of PP in a student classroom depend on the instructors skills and motivation toward PP. Brought10 [14] mentions that if a pair consists of two students with different experience levels than the weaker one will not benefit from the stronger one because he cannot follow and understand his problem solving approach. In Salleh et al.'s meta study [56] the conclusion was that, the productivity of pairs is similar or better of pairs and that there are no negative effects to the academic performance.

Coder motivation Hanks [26] paper focuses on the attitude of students towards PP, that participated in a PP study. In total 115 students participated in the study and were taught by three instructors. By random the students were split into two groups: one group performed PP and in the other one the students used a tool that supports distributed-pair-programming. Afterwards the students answered surveys to evaluate their attitude. Surprisingly the results differ depending on the instructor that held the course. In two classes the students significantly liked PP, would do it again and think they learned more because of PP (6/7 points). In the other class it received an average rating (4/7). The authors remark: "If instructors using these techniques cause students to have negative attitudes, it is possible that this would prevent the techniques from being successful in general."
The results also indicate, that there has been no performance difference between PP and distributed-PP.
Opinion: This paper was chosen to show that the success of PP also depends on pedagogical

techniques. If the instructor does not teach principles correctly or may have a negative attitude towards PP himself it may affect the students or professionals attitude as well.

Begel and Nagappan [10] reported on a large scale survey at Microsoft to evaluate the use-distribution of PP in their company. 10% of the software engineers at Microsoft received the survey (chosen by random). 22% state that they have used Pair-Programming so far, but only 3.5% in the current project. The significant majority (64.4%) state that PP performed well for them and that the use of PP produces higher code quality (65.4%). However, this majority decreases to 40% when the team size or organizational level increases.
The results about if PP means more effort for a project are divided: 37.5% agree and 25.4% disagree.
64.5% believe that PP leads to fewer bugs in the code.
Most of the developers would like a PP partner that has complementary skills to their own and who is flexible and communicative.
Opinion: This paper is only a survey about how PP was used at Microsoft and on the developers opinion towards it. There is no information how PP actually worked out in the projects. The paper still shows that PP is accepted by the majority of professional developers who believe in the positive aspects of PP.

Summary Nosek98 [47], Williams00 [61], Cockburn01 [15] and Williams03 [62] report of an increase of the developers motivation due the use of Pair-Programming. Phongpaibul06 [53] said that the increase of productivity when using PP can result from the Thai' culture, since they like socializing more than Europeans or Americans and thus it motivates them. Lui06 [35] mentioned that expert-pairs bring a lower boost than novice-pairs because the experts are more likely to get distracted or bored while navigating since they already know if one programming approach will be successful. Müller and Padberg conclude, that pairs with a high "feelgood factor" achieve better results than pairs that don't feel comfortable with PP. [43] Also the personality of the pairing programmers play a role if a programming pair can work together efficiently or not. [28]

Distributed Pair-Programming Hanks published papers [26] [27] stating that distributed Pair-Programming (the two pair members are not at the same physical location and collaborate over internet) can achieve the same results as the standard Pair-Programming.

Several other studies about distributed Pair-Programming exist but they are of no further importance for this thesis because we will concentrate on non-distributed Pair-Programming to lower the number of variables.

2.3 Coding Contests

Compared to Pair-Programming the number of studies made about coding contests is limited. In this subsection research about coding contests is presented and discussed.

In 1993, Astrachan et al. [4] analyzed existing programming contests and presented a contest rationale to provide a model that will increase the interest in programming as an essential aspect of computer science.

Ernst et al. [22] presented in 1997 strategies and tactics on how to compete better in programming contests. They discussed which groups of problems occur over and over again at contests. Two quotes of the paper that are related to PP: "It is our experience that the most efficient way to write a program is to write it alone. In that way you avoid communication overhead and the confusion caused by differing programming styles." and "We found out that two people examining hard problems often lead to creative solutions."

Manzoor [38] focused on the ACM ICPC contest and two online judges that can be used for training, namely the University of Valladolid online judge [1] and the USU online judge. In this paper from 2001 the author explained common mistakes that happened in the C or Pascal solutions and gives suggestions on technical problems.

In 2002, Shilov and Yi [57] discussed the decreasing motivation of students to consider mathematical foundations of formal methods, especially undergraduate students. The authors suggest that using coding contests, which are considered to be fun and motivating, can show the students that the tasks needed to be solved there could be addressed easier if they would have a fundamental knowledge of formal methods. Citations: "Training sessions are good opportunities to present students with challenging programming problems that cannot be solved without theoretical background in spite of simple formulation. The trainers should provide students with background theory as soon as students realize the programming complexity of these problems." and "We intend to inspire readers from academia and industry to consider the importance of a popular presentation of theoretical research for better computer science education."

Andrianoff et al. [2] criticized in 2004, that the traditional ACM contests did not force the participants to program in an object oriented way or anticipate a minimum of coding quality. They suggested that instead of the classic tasks which have to be developed each time from zero, the problems should be about existing classes that have to be extended or improved. This way the participants are forced to program object oriented. They performed an experiments with 350 high school students with positive results.

Patterson [50] discusses ICPC contests about the increasing number of participants and the improving performance of China and Russia compared to the declining demonstration of students from USA. He criticizes, that the Russian champions get invited to the Kremlin to meet their president while in the U.S. this is done with football champions.

In another paper, Manzoor [39] analyzed the submission database of Valladolid Online Judge (UVaOJ) to point out, that this data exists and can be used by researchers or programming contest organizers. This online judge contains a large collection of programming problems which can be tried to solve to gain practice. Any person can browse the problems at any time and submit solutions, i.e. source code, which will be run by the judge-program which answers with a return code to see if one passed the task or an error occured, e.g. Accepted, Wrong Answer, Compile Error, Memory Limit Exceeded, etc. Manzoor used this return codes to analyze error rates of programmers and if they improved their performance the more they practiced. Another part of his paper was to review which countries use UVaOJ the most and if it correlates with ICPC final-qualifications or IOI performance of these countries.

Boersen and Phillips [13] presented programming contests hosted in New Zealand. This contests focus on female high school students as participants to motivate them to try out pro-

[1] uva.onlinejudge.org

gramming "(...) without the possible distraction of more dominating males.". [13]

Cormack et al. [18] discusses the most known contests like the ICPC, IOI and TopCoder and how they could be improved to give a rewarding experience to all competitors. They illustrate different contest designs and their elements, like the scoring-system, test cases, etc. and discuss their advantages and disadvantages.

Amraii [1] analyzed team strategies used by programming contests, especially the ICPC. He describes how to prepare or learn, how to form the team and how to distribute the tasks between the team members.

2.4 Summary

A large number of researches and many experiments had been made on the effects of Pair-Programming. Although, most of the presented experiments (5 of 8) were conducted with students as subjects while the other three studies were made with professional programmers. These studies vary from effects on performance to quality, learning and motivation. More experiments with professionals were requested by researchers to bring more evidence of the effects of Pair-Programming (PP). [3] [48] Another problem so far with many experiments made about PP is, that the subject size was small. The effort to host an experiment with a large number of professionals, i.e. hundreds and more, is too high compared to the budget of most university institutes. These professional coders also have to be motivated to participate in the study and apply to its rules. If this is done by monetary compensation the effort will rise even more.

On the other hand there are programming contests with thousands of participants, including professionals but the designs are very restricted (fixed team size, bound to institutions, etc.) that do not allow heterogeneous groups like solos, pairs or teams. Research on coding contests is limited, many studies focus on strategies (how to compete better) [39] [1] [38], while others suggest to use coding competitions as motivators for learning reasons [13] [18] [57].

There have been studies on how participants competed at programming tasks [39] and [22] but this has only been done on ACM's ICPC contest or the Valladolid Online Judge trainer. There is a gap of research about using large scale coding contests to test software development methodologies. The Catalysts Coding Contest (CCC) can provide the necessary information since it is allowed to participate alone, or in teams of two and three as well because there are no restrictions to universities (ICPC), age (IOI) or experience. Using the result data of a coding contest, in particular the CCC, would allow it, to compare teams against individuals and juniors against professionals in performance and quality aspects in a dimension that would be not effortable for universities. (For a detailed description of the CCC please see chapter 5.) The prices like money, fame or job positions offered when achieving good results in coding contests motivates each participant to give their best.

Performing a study based on the results of coding contests could also motivate contest organizers to publish their data sets or adapting the competition mode to give researchers a wider scientific access.

CHAPTER 3

Research Issues

In this chapter we will introduce the main research topics. Each topic contains two research issues (RI).

3.1 Coding contests as vehicle for empirical research (Research Topic 1)

The first research issue describes experiments in software engineering. Research issue 1 is about how coding contests can be used as a host for experiments, while the RI2 describes more generally the scientific benefits.

3.1.1 How can coding contests support controlled experiments? (RI1)

Coding contests are events where programmers compete against each other and try to solve given problems as fast as possible. (See chapter 2.3 for related literature and examples of coding contests.) Organizers of coding contests want to attract skilled coders and provide a challenging and fair setting.

When researchers conduct controlled experiments, their subject group usually consists of students only [64] and is spread over a whole semester on various homework assignments/exams during a programming course (E.g. Cockburn and Williams 2001 [15]).

This research issue aims at combining coding contests and controlled experiments and describing the experience of a pilot experiment embedded in a coding contest, in particular the Catalysts Coding Contest (CCC). This can bring advantages for the researcher as well as the contest organizer.

The contest organizer benefits from a) more attention that his/her coding contest gets (e.g. professors advertising the contest to students) and b) that a contest can be advertised as "science-supportive". I.e. many contests have the reputation of being 'only' a recruiting event of industry companies, but a contest that also supports science can attract programmers that have no interest in programming contests.

The large number of participants and the fact that a researcher is exempted from organizing an experiment all by him/herself are the big advantages for researchers when embedding their experiment into coding contests. This can save time and budget for the researcher.
Coding contests and controlled experiments both require to be held in controlled environments. This fits to coding contests because the final rounds are usually held offline in a controlled environment.
Another advantage is that contests attract not only students but also professional programmers who already gained practical experience. Having students and professionals as subjects in the same experiment is necessary when conclusions in a general context want to be made.

Since the reviewed contests of chapter 2.3 are held regularly they allow replications of previous studies' results. Findings of experiments can be easily verified or questioned if the experiment is repeated in the following year.

RI3 will accumulate the knowledge gained by the coding contest pilot experiment and emphasize the following points:

- Show that the **level-based concept** can be used to compare individual- and pair-programmers during the same experiment.
- Present the **large, heterogeneous subject group**
- Show that experiments can be **embedded** into coding contests **and repeated**.

3.1.2 How can the concept of coding contests support research and replication in the field of empirical software engineering? (RI2)

The second issue of the first research topic discusses how coding contests can support science in the field of empirical software engineering. E.g. if it is useful for empirical-ecosystems.

Biffl et al. [11] suggest to use empirical ecosystems to easier share information of experiments between different researchers. (See also chapter 2.1)

Like discussed in RI3, embedding experiments into coding contest reduces the necessary effort needed by a researcher when planing to conduct large-scale experiments, because the contest organizer takes the work of hosting the experiment out of the researchers hands. Thus, the researcher can concentrate on his/her research works and thereby the coding contents indirectly contributes to a body of knowledge by publishing new results or verifying old results faster.

This research issue describes what coding contests, in particular the CCC, can contribute to a body of knowledge. I.e. how can the CCC be used to enhance the knowledge of topics under investigation? Several assumptions are already described above. In chapter 7 we will discuss if the assumptions can be reconfirmed.

3.2 Empirical evaluation of software-development-techniques effects in context of coding contests (Research Topic 2)

The second topic hold a number of hypotheses that will be tested using the coding contests results. The outcome of the hypotheses testing will be used to argue for or against the research

issues.

RI3 concentrates on the effects of Pair-Programming, while RI4 focuses on the effects of experience on software quality.

3.2.1 What are the effects of developing software in teams? (RI3)

Studies about effects of Pair-Programming naturally require pairs (experimentation group) and individual coders on the other side (control group). The contests, presented in chapter 2 only allow either teams or individuals to participate, with the exception of the CCC. Allowing coders to attend alone, e.g. having solos and teams of three at the ICPC, would adversely affect them in the contest because they still have the same amount of tasks like groups. However, a level based design where tasks have to be completed successively and information is only present for the current task would eliminate this problem since team members that are not sitting in front of the shared computer cannot solve other tasks on the paper in advance.

For this research issue the level-based concept will be applied on the CCC to compare the results between pairs and individuals regarding duration, effort and defects. Further, each level requires predefined software skills in order to solve it (efficiently).

It is reasonable that duration and effort will be evaluated for different hypotheses because if Pair-Programming needs less duration but higher effort, it has still its advantages like debated by Padberg and Müller [48].

In chapter 2 is a summary of related studies that also published the results of their experiments. On average, the time needed to complete a task by using Pair-Programming was 41 % lower than for single programming. The average effort of pair programmers was 19 % higher and the average number of defects made was 18 % lower.

According to the meta study presented by Hannay et al. [29] it is expected that the results of this experiment will confirm the hypotheses H3.1.1 (pairs are faster) and H3.3.1 (pairs make fewer defects). However, their meta study showed that pairs need more effort than individuals, so it is expected that H3.2.1 will not be confirmed.

Still, the alternative hypotheses are formulated in a way that a positive effect is expected if Pair-Programming is used. If the experiment results will reject a null hypothesis but there is a negative effect when using Pair-Programming than these findings will be debated as well.

Null and alternative hypotheses for research issue 3:

- **Duration** Duration is the time it requires to complete a level (task).

  H3.1.0: There is no significant difference in time needed to complete tasks between pairs and individuals.

  H3.1.1: Pairs significantly solve tasks faster than individuals.

- **Effort** Represents the effort that was needed to complete a level (task), i.e. man-hours.

  H3.2.0: There is no significant difference in respect to effort needed to complete a task between individuals and pairs.

  H3.2.1: Pairs significantly require less effort than individuals.

- **Defects** Defects are failed acceptance test cases that are run when a solution is submitted.

  H3.3.0: There is no significant difference regarding the number of defects between pairs and individuals.

  H3.3.1: Pairs significantly produce fewer defects than individuals.

3.2.2 What are the effects of experience on coding efficiency in the context of programming contests? (RI4)

Programming contests are especially popular for very skilled and experienced coders since they know they have the knowledge and confidence to be successful. Unexperienced programmers usually drop out in the qualification rounds of the events. The CCC is small enough to allow newbies to attend in the same round as the experts which means that their results can be compared. Famous coding contests are usually too hard for unexperienced coders. A contest that wants to attract people with different levels of experience has to give all of them a challenge. The level based design allows this, because the first levels can be easy (but still challenging for newbies), and the later levels more difficult (even for the experts). A contest should be motivating for all participants. The results of the CCC will be evaluated to find out if juniors and professionals are attracted by the contest at a similar level.

Using the same level-based approach like for RI3 the focus lays on how participants with much experience compete against ones with less experience in respect to time needed to complete tasks and the number of produced defects.

It is expected that experience helps to complete tasks faster and that less defects are produced.

There will also be an evaluation how pairs with different experiences compete against other pairs, as well as individuals against other individuals. According to Lui and Chan [35] juniors profit more from Pair-Programming than professionals. Hannay et al.'s meta study [29] confirms Lui's and Chan's assumptions.

Null and alternative hypotheses for research issue 4:

- **Duration** Duration is the time it requires to complete a level (task).

  H4.1.0: There is no significant difference of time needed to complete tasks between high and less experienced participants.

  H4.1.1: High experienced participants complete tasks significantly faster than less experienced participants.

- **Defects** Defects are failed acceptance test cases that are run when a solution is submitted.

  H4.2.0: There is no significant difference regarding produced defects between high experienced and less experienced participants.

  H4.2.1: High experienced participants produce significantly fewer defects than less experienced participants.

CHAPTER 4

Research Approach

4.1 Research approach

In chapter 2 we discussed the related work including research gaps (based on a literature review),and in chapter 3 the research issues for this master thesis. The plan was to use a coding contest with an embedded experiment to show that this approach can be useful for researchers in the field of computer science. This will be done by answering RQ 1, which focus on the general approach, and RQ 2, which is a proof of concept that an embedded experiment can provide useful information for open research issues and thus support science.

After the literature review and defining the research issues, and experiment is planed and executed. The second step is to plan and conduct the experiment using the process described by Wohlin et al. [64] The experiment was split into well-defined steps (Planning, Description, Operation and Result-Extraction) to ensure authenticity of the data and minimize threads to validity (necessary for RQ 1).

The experiences gained by conducting the experiment and evaluating the results will be used to discuss RQ 2.

Subsequently the results of the experiment will be presented and discussed (RQ3 and RQ4). For each hypothesis the according data sets will be selected and tested using IBM's SPSS statistic tool. More precisely, the means and variances of duration and defect values will be calculated and compared using descriptive statistics as well as analysis of variance (ANOVA) tools like the Student-T-Test.

4.1.1 Embedded experiments in coding contests

Coding contests exist in various dimensions like Google's Codejam with thousands of contestants worldwide including multiple qualification rounds to very small events for just a handful of programmers (e.g. inside a small company or school class). Regardless of the size, they usually share the competitive and controlled environments. For a contest there have to be winning prizes

that motivate the participants to try their best (competitive) but on the other hand, there have to be rules to provide fairness. See chapter 1.5.1 for a list of coding contests.

Many requirements for scientific experiments overlap with the features provided by coding contests (E.g. controlled environments). Other features of contests come in very handy, like the high participants number.

Our idea was to combine them by embedding an experiment into a coding contest. This can be done without the knowledge of the participants.

In the next subchapter 4.2 the experiment is planed in more detail.

4.2 Evaluation concept

This section describes the general conditions that are a necessary base for our research issues, the methodological approach for the experiment and the selected coding contest that will act as our "vehicle".

4.2.1 General conditions for the experiment

The following conditions are needed to be fulfilled by a coding contest in order to be useful for our experiment.

- **Team size.** RI3 focuses on Pair-Programming (PP), thus the experiment has to be made with subjects that are teamed up into pairs as well as single participants for comparison. PP requires that both programmers of a pair are sharing one machine and focusing on the same tasks.
- **Profession.** For RI4 the results of participants with much experience in the field of programming (professionals) will be compared to those of juniors. I.e. The participants pool have to contain programmers with either high or low experience in software engineering.
- **Controlled environment.** The experiment has to be conducted in a controlled environment. I.e. participants are seated in physical locations where the process can be overwatched and cross-team-collaboration should be prevented.

4.2.2 Methodological Concept

After evaluating general conditions for the planned experiment, discussing the idea of embedded experiments and presenting the upcoming coding-contest-host for our experiment, we will discuss the further necessary methodological steps.

1. Plan and execute the experiment

a Study the Wohlin process [64]. Before starting conducting an experiment, we needed to have a robust understanding of the general process and pitfalls one can make. Since we didn't perform a scientific, empirical experiment before we have to study the process, using the book of Wohlin et al.

b Planning of the experiment. Formulate the planning of the experiment in details applying Wohlin et al's process. This also includes the preparation of the material, i.e. the programming tasks for the participants.

c Observe execution and collect data. To ensure the validity of the data we will observe the experiment execution procedure on site. A local contact with the participants also helps to gain a faster understanding of the subject distribution (since we don't know these for sure before the contest).

d Data extraction and validation. After the contest finishes, extract the data using the exporting tools from Catalysts. Transpose the data into the statistical tool that will be chosen for the descriptive statistics and ANOVA tests. Ensure that no errors by were made in the data extraction and transformation.

2. Analysis of the data

a Data preparation. Describe the collected data in general including descriptive statistics. Remove invalid data points from the statistic.

b Tests hypotheses. Aggregate the data for the needs of each hypothesis and perform the T-tests to test them and describe the results.

3. Interpretation

a Discussion of the approach. Discuss the approach of embedding experiments into coding contests like described in RQ 1 and answer the research issues.

b Proof of concept discussion. Discuss the results of the hypotheses of RQ 2.

4. Conclusion and future work

4.2.3 Catalysts Coding Contest

In collaboration with Catalysts GmbH the Catalysts-Coding-Contest (CCC) was chosen for the embedding of the experiment. The setup of the CCC fulfills all the requirements listed in chapter 4.2.1.

Figure 4.1 shows the registration numbers of the previous CCCs in the past years. According to these numbers the CCC is not a small contest and thus will produce many valuable data points. A high number of participants is the key motivation to embed experiments into a coding contest.

Allowing both, teams of individual programmers, to attend and compete against each other at a contest also need a setup or mechanism to make it fair for individuals. Large contests like ACM's ICPC allow teams of three with only one workstation. This usually leads to one person who is programming and two persons who solve the upcoming problems and tasks on paper before they become coded. Such a paralleled setup would handicap one-person-teams.
The CCC counters this problem by using a level based contest approach instead of giving each participant a set of multiple tasks where one can choose the ordering him/herself or solve them in parallel. Level based means that only one task/problem is visible for each team/individual at

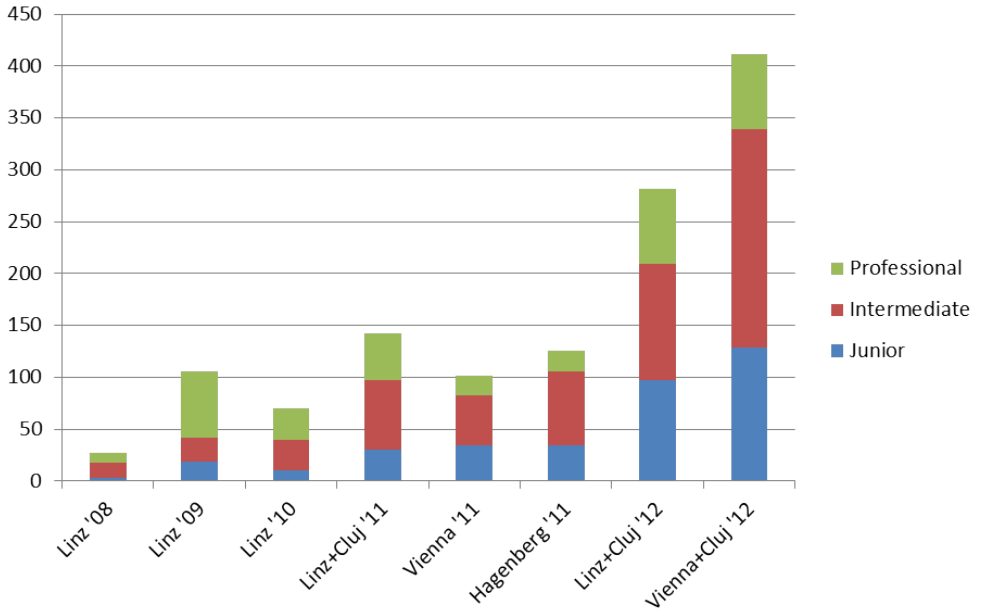


Figure 4.1: Trend of the registration numbers for the CCC. These numbers also include online-only participants.

a time. This task has to be solved in order to get the next task or problem. Aim is to solve as much tasks as possible. Thus, the level based approach gives teams or individual programmers even chances.

The CCC already has a working environment that allows full logging and monitoring of the action of the participants. Each try to submit a solution is stored in their database and also the source code can be uploaded. At the end of the contest the results can be easily exported to a spreadsheet file.
Before the contest starts the participants have to register at a front desk and provide their knowledge level. During the contest Catalysts has supervisors patrol the workstations to give support if problems occur and to prevent cheating.

A detailed description of the CCC is given in chapter 5.1.

CHAPTER 5

Experiment Description

This chapter describes the aspects of our experiment integrated in the Catalysts Coding Contest (CCC).

5.1 Coding Contest Description

General information The Catalysts Coding Contest (CCC) is a contest held by Catalysts GmbH, seated in Linz, Austria, where programmers of all ages and experience levels can measure themselves against each other by trying to solve a level based problem set as fast as possible.

The coding contest is structured into four steps (see Fig. 5.1): (1) Contest Preparation which includes setup of the material, advertisements, and organizational issues; (2) Registration of Participants, i.e., individuals and teams (pairs); (3) Contest Execution & Data Collection; and (4) Data Analysis and Award Ceremony. The steps are further explained in details in the sections below.
Figure 5.1 is taken out of our XP-2013 paper [63].

There are no age or experience restrictions for the participants, further the team size can be chosen freely (one to three persons). This makes the contest very heterogeneous and combined with the high number of participating groups (150-200 in Dec 2011 and April 2012, 300 in October 2012) a researcher can use it to conduct large-scale empirical studies.

A few weeks after each contest, Catalysts publishes the anonymized content data on open-data-servers to let the research community profit from it.

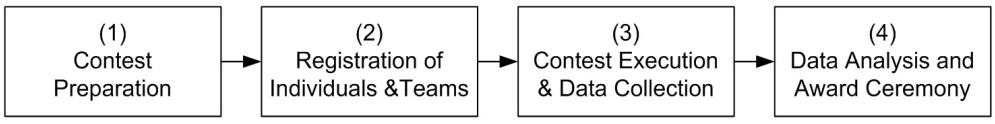


Figure 5.1: CCC Process Steps [63]

Multilevel system Using a multilevel system is the most important difference between the CCC and most other programing contests, which describe a set of non-relating problems and whose participants can choose the order in which they want to solve them. At the CCC a team can only solve one problem at a time. Only when a level is completed, a team can move into the next level which usually is either an additional module for the previous task or demands a more generic approach. This usually means that one has to build a flexible program because there is a high change that it has to be re-factored in the following levels.

Aim of the contest is to solve as many levels as fast as possible. For each level the participant gets a task description, some example test cases (TCs) and a group of more complex TCs (acceptance test). The acceptance test cases need to be solved to successfully finish the level. An acceptance test is an input-output-value pair where the output is hidden to the participant. The contestants' program shall produce the output for a given input and submit it to the contest server which replies if the solution is correct or not correct. The participant can submit the solution for each TC separately to see if it works and once all of them are correct he/she will reach the next level and get the next assignment and TCs. To prevent brute force solutions the participants will get a one minute penalty on its total time of each failing test case. If one fails a test case multiple times he will get the penalty for each time the solution is rejected. To increase the experiment value for researchers the participants will receive a two minutes bonus if they upload their source code for the current level.

Contest procedure Fig. 5.2 shows the procedure of the level based design during the contest by focusing on the first two levels. A quiz master (i.e. an application on a webserver called Catcoder) is distributing task descriptions individually to each participant. Step one in the figure shows Catcoder handing over the task description for level one to each participant. Once they have their task, they can work on them and submit results back to Catcoder using a web interface (step 2). The results are then checked and the recipient receives a feedback whether it is correct or not. No more information is given than correct/not correct.
When all subtasks of one level are correct, then step 3a applies: the participant can move on to the next level which means they get the next task description.
When a submission is not correct (step 3b), then the participant can try again as often as he/she likes. However, there is a malus (in form of bonus time) for each wrong answer.
The contest is finished after the complete time ran out or all teams completed all tasks. [63]

After the contest, the ranking of the participants will be ordered by their adjusted, total time. The first 30 places will receive price money from Catalysts and theirs sponsors to make the contest more attractive. This is unusual for academic researches but has the advantage of motivating professionals that have no connection to a university to participate.

Times and locations Catalysts organizes three CCCs per year: Linz in April, Vienna in October and Hagenberg in December. Simultaneously with the CCC in Linz and Vienna a twin event is held in Cluj, Romania. Since the beginning of the CCC the average number of participants increased from around 20 to 300.

Figure 5.3 shows a screenshot of the current Catcoder version and how a participant uses the test cases.

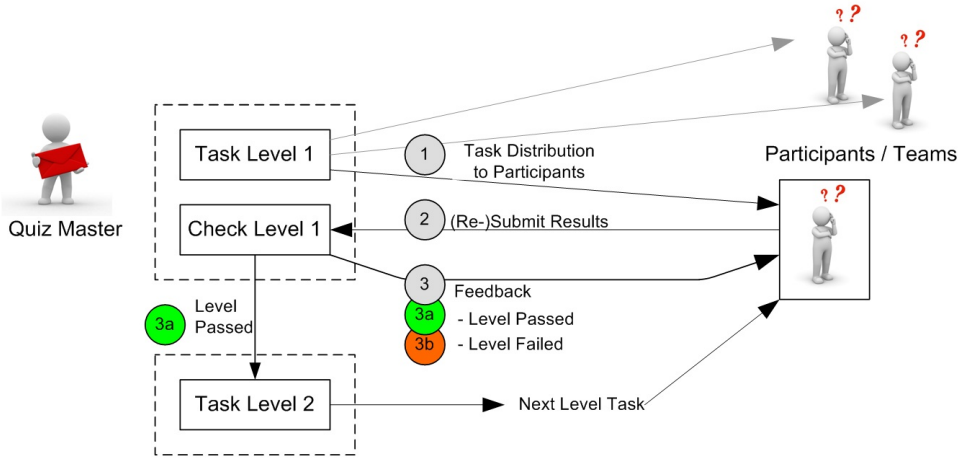


Figure 5.2: Level Design of the CCC, showing the process how to finish level 1 and advance to level 2. The quiz master is the Catcoder server that can be interacted with via browser. Figure is taken over from [63].

5.2 Experiment definition

The experiment is defined according to Wohlins [64] "Experimentation in Software Engineering". The purpose of this section is to properly lay down the intention of the experiment.

Object of study The objects studied are the development strategies solo- and pair-programming and the developers experience.

Purpose The purpose is to evaluate the efficiency and monitor the development process of different programming strategies and programmer experience.

Quality focus The quality focus is the performance of development strategies and experience while solving programmatic problems.

Perspective The perspective is from the researcher's point of view.

Context The subjects of the experiment are programmers from high schools, universities and industry. They attend in teams or as single programmers. The experiment is held as a blocked subject-object study.

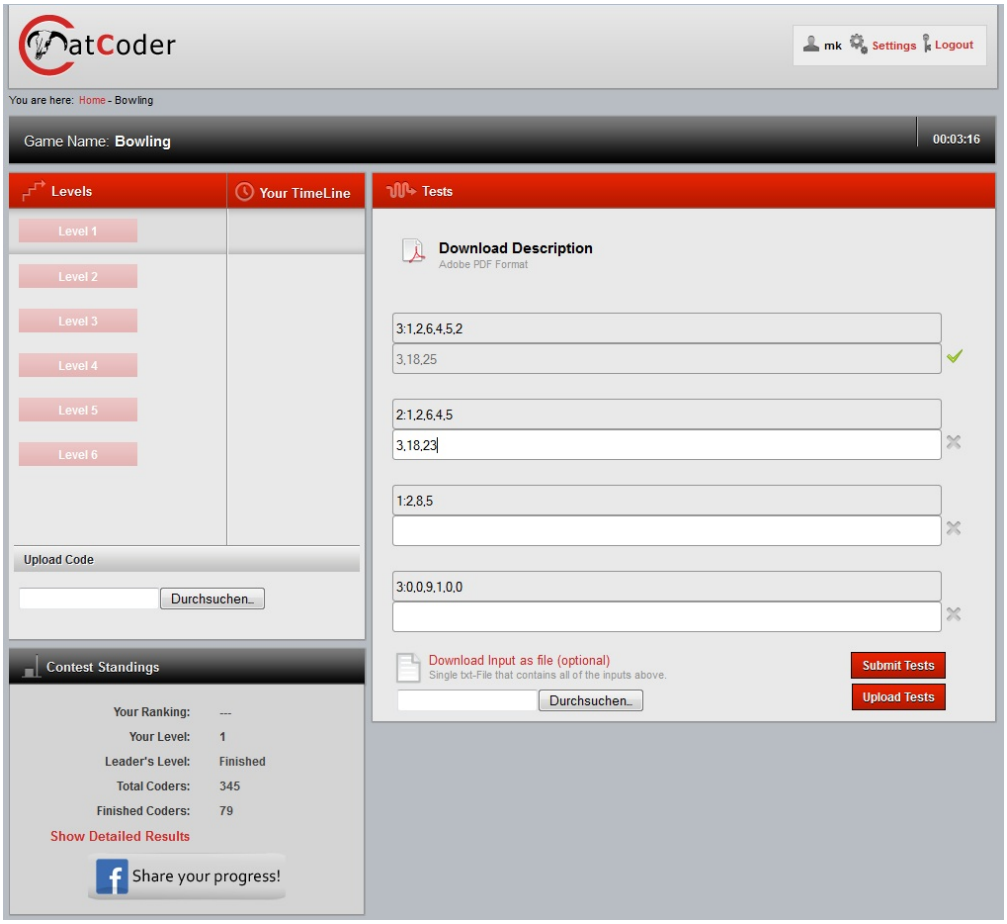


Figure 5.3: A screenshot of the Catcoder application showing the level structure on the left, a link to download the task description on the top and below it, the test cases for the current level. Test case 1 has been successfully commited already, test case 2 has only been filled out but not submitted and the other test cases have not been tried yet.

5.3 Experiment planning

According to Wohlin, this section is split into the following subsections: Context selection, Hypothesis formulation, Experiment design and Instrumentation.

5.3.1 Context selection

The experiment will be held in the context of the Catalysts coding contest (CCC). The concept of the CCC has been proved multiple times over the years. Using the existing contest, a researcher can use it to conduct an experiment without having to search for participants or telling them that it actually is an experiment. For this diploma thesis' experiment, minimal changes in the CCC process were required.
Catalysts already provides the following:

- **Location** Catalysts chooses the location that fits the expected numbers of participants. In the past this had been seminar rooms of universities, offices in industry buildings and public locations like the Vienna City Hall or the Castle of Linz.
  For this experiment the contest was held on two locations simultaneously: In the City Hall of Vienna[1] and in the Universitatea Babes-Bolyai[2] in Cluj. Both locations had the same setup/assignments and started/finished at the same time.
  There is also the possiblility to participate via internet, but these online contestants were ignored for the experiment because they didn't work in a controlled environment.

- **Infrastructure** The participants have to bring their own workstations, while Catalysts provides the network infrastructure, i.e. routers/hubs, network cables, internet connectivity and a contest server. Tables and chairs are supplied by the locations' owner.

- **Level design** The level design exists since the first CCC held in 2007. The number of levels varies, depending on the contest duration and program complexity. The level descriptions for the experiment were created in collaboration between us and Catalysts. For this experiment a seven-level based problem was chosen.

- **Level complexity** The number of levels and their complexity is designed in a way to be challenging even for excellent programmers, due to the limited time. Catalysts approach for the difficulty is: Starting levels (level 1-2) are easy so that every programmer can solve it, while the difficulty increases and only the best 5 % should be able to solve all levels in time. The problems don't require surpassing algorithm knowledge or math skills like other coding contests but focus on efficiency and re-usability. Refactoring a previous levels' solution, because the requirement in the higher level became more complex, is very common in CCCs and fits well to the daily routine of a software engineer. For the detailed descriptions of the task see section 5.6.

[1] http://www.wien.gv.at/english/cityhall/
[2] http://www.ubbcluj.ro/en/

- **Participants** There is no limitation for programmers to participate except Catalysts employees and a total maximum (300 in this experiment) due to room size restrictions. Everyone who registers on the webpage and accepts Catalysts' terms of participation can join. Experience showed that the biggest group is professionals (master students and young practitioners), followed by juniors(pupils and bachelor students) and professionals (programmers from the industry with many years of experience).

- **Team sizes** Team size can be chosen freely by the participants. The size varies from one to three persons per team.

5.3.2 Hypothesis formulation

The hypotheses are formulated in chapter 3.

5.3.3 Variables selection

In this subsection all variables needed for hypothesis testing are described. Each variable has a name, a measurement scale and a description.

Independent variables can be chosen and controlled by the researchers and stay fixed for the whole experiment. Dependent variables cannot be influenced by the researcher and are measured during and after the experiment. Treatments are the methodologies under test that influence the dependent variables.

Independent variables

- **Level concept** The tasks are split into seven levels that have to be solved in a specific order.

- **Task descriptions** Each level has a task description that explains how it can be solved.

- **Test cases per level** For each level there is a number of test cases which have to deliver the correct outcome.

Dependent variables

- **Working time per level** *Ratio* Shows how much time was needed to solve level x.

- **Total levels solved** *Ratio* Describes how many levels each team solved in total.

- **Number of defects per level** *Ratio* Describes how many test cases failed before the level was solved.

- **Total defects** *Ratio* Describes how many test cases failed in total.

Treatments

- **Time size** *Nominal* Tells if subjects participated alone or in a team.
- **Experience** *Ordinal* Splits the experience of the participants into: Junior, Intermediate and Professional.

5.3.4 Experiment design

Randomization Like described in the subsection context-selection (5.3.1), the CCC is an industry contest where software engineers can freely choose to participate, without restrictions on their experience level or programming language.
Unlike other experiments where there are limited resources, the two facts (a) random team size and (b) random experience, combined with the high number of contestants, allow it to achieve the necessary statistical randomization of subjects without further selection. It is expected to obtain a representative sample of the software engineer population.

5.3.5 Instrumentation

"The overall goal of the instrumentation is to provide means for performing the experiment and to monitor it, without affecting the control of the experiment." by Wohlin et al. [64]

The following instruments are used to perform the experiment:

Level description Each level has a task description which explains the aim of the level. One to three small test cases, depending on how complex the task is, are included in each description to give an example of what the levels' implementation should result.

Test cases For each level there is a number of test cases (4-8) that have to return the correct value to successfully finish a level. A test case is always an input-output value pair, defining which value the program shall output if you insert an input value. The input values are public for each level, the output values are only known by the operators and the server application that checks the results of the contestants' programs.

Contestant data collection Each contestant has to register on the CCC homepage and fill out basic personal data. This data is checked when the programmer appears on site of the experiment at the welcome desk. For further research, this information is needed to know the team size and experience level of the participants.

Program data collection The server application "Catcoder" manages the advance of each contestant. Each time a participant submits an output (solution) of a test case to the server, the action and its results are logged.
After the contest the server files can be used for further analysis.

5.4 Experiment operation

This section describes in detail the experimentation steps: preparation, execution and validation.

5.4.1 Preparation

Participant motivation Months before the contest, the advertisement process for the contests starts. Universities and schools are asked to promote the CCC to their students, Infos on discussion boards in the software engineering portals are written and befriended companies invited to send their employees for a challenge.
The high number of participants will equalize the various backgrounds. Further it enables to select blocks (groups) and still have enough subjects to make empirical analyses on them.
Especially professionals are very sensitive about their result data because it can cast a shadow on their working career if it is made public that they performed poorly. Consequently all personal data is handled very carefully and will not be published.
Since the contest is basically an industry contest, the price money is needed as a motivation factor to attract software engineers to attend. Effects on the experiment outcome will be minimal since the participants don't know the aim of the experiment. The idea is, that the price money will motivate all teams to give their best and that fits to the aim of the experiment: evaluating the efficiency of the various treatments.

Instrument preparation Task descriptions and test cases are elaborated in collaboration between Catalysts and us. Two weeks before a contest, internal tests are run to test if the complexity fits the time limit. Experienced Catalysts employees are trying out the contest in a real environment and give feedback if the descriptions are clear or ambiguous. On the other side the task creators evaluate the progress of the test participants and add/remove/adapt levels. This test is repeated one week before the contest to check out the adjustments.
The task descriptions and test cases are distributed by a Catalysts application called Catcoder. This server has to be set up at the experiment location and will be accessed via LAN.
Before the experiment can start, the infrastructure has to be build up: The participants will have to bring their own workstations but Catalysts provides the LAN infrastructure. Tables and seats, switches and network cables as well as a server machine has to be setup. All the participants will have to prepare is to register at the welcome desk, build up their workplace, connect via browser to Catcoder and register an account. At least two persons are necessary to take care of the participants at the welcome desk to check their personal data.

5.4.2 Execution

Environment and duration Like further described in section 5.3.1 the experiment was executed in the City Hall of Vienna, Austria and the Universitatea Babes-Bolyai in Cluj, Romania on the 19th of October 2012. The execution time was 14:00 until 18:00. The participants were able to register and build up their workstations from 12:00.
On both locations Catalysts employees operated the contest, while we were present at the location in Vienna.

Process Each participant had to register before the contest at the registration desk. Afterwards they can choose their place to build up their workstation, connect to the LAN and create an account at the Catcoder server. This account is the connection between a team an the Catcoder. The server saves the current status of each 'user' and supplies the teams with the current level descriptions and test cases. Until 14:00 the participants can download the encrypted version of the first levels assignment.
Before the contest starts the rules for the CCC are explained to all participants.
Once the rules are explained the key for the encrypted description of level 1 is published and the teams can decrypt the first assignment. This system is used to prevent network bottlenecks at the beginning of the contest and allows a fair start for all participants.
At 18:00 the Catcoder does not accept any new solutions for test cases and a ranking is generated. The winning ceremony is held using this ranking.

Data collection The personal data, i.e. experience and team size is collected in advance of the contest via web forms. This data is checked at the registration desk, but it is still possible to receive wrong or incomplete information.
Contest data is collected by Catcoder during the execution. "Communication data is captured in a log file including submitted results, test cases, timestamp, user name and test case values. In addition participants are asked to submit their source code in a designated folder structure. Note that source code submission is voluntary. Catcoder, the server application that hosts the contest, extracts the information of the log file and generates a human-readable spreadsheet presentation including a list of participants ordered by rank. In addition registration data, experience levels and years of experience (a control value) is available in a database log file for further usage." taken over from the paper presented at the XP 2013 [63]

Data Analysis "The final result set (i.e., a spreadsheet) holds data of individual participants, experience data, team information (solo or Pair-Programming), and required duration and defects per level. Note that we used the data in an anonymous way by removing confident and private data for evaluation. For evaluation purposes we removed participants who did not provide the requested data or did not permit further data analysis (see Section 4.2 for details). We applied descriptive statistics for data analysis and the t-test at a significance level of 95 % for hypothesis testing." taken over from the paper presented at the XP 2013 [63]

5.4.3 Validation

The personal data has to be validated after the contest to ensure its validity. Unfortunately, it cannot be ensured that all data provided by the participants is correct. This can affect the dependent variables because if a user gave false information than his/her data set has to be invalidated and removed from the study. Since the Catcoder functionality has been proved several times it can be ensured that 'cheating' is not possible and that the collected contest data is accurate.

5.5 Participants

We want to have all kinds of experience levels (junior, intermediate, professional) in the experiment and the CCC fulfills this request by having no participation restrictions, e.g. age, university or organization. Since it is a coding contest, the participants will attend by their own intentions and will not be selected by us or Catalysts.

The participants will also form teams by their own (Teamsize: 1-3 persons). Catalysts wants to allow teams of three in their contest, these teams will be ignored for our experiment because we are only interested in individuals and pairs. In this approach there is the disadvantage, that the participants will not be equally divided up by team size and experience level. E.g. all single programmers could be professionals and all pairs could be juniors. However, in previous CCC's the groups were rather equally divided.

5.6 Material

Study material includes (a) technical infrastructure and (b) contest material, i.e., task description, requirements and test cases:

Technical infrastructure It was in the participants own responsibility to bring their own workstations with their local software environments installed. Since the programming language and the development environment were not restricted, the participants were able to work in their familiar language and environment. Catalysts provided internet connectivity (via LAN cable) to connect to the contest web server and to allow the participants to access language APIs.
The participants had to use their browsers to connect to the webserver and download the task descriptions as PDF files. The descriptions lay in password encrypted zip files which had to be unpacked.
The webserver was hosted by Catalysts using their own application that acts as the quiz master. [63]

Task descriptions The contests main task was to write a simplified lip reading program, that receives as input a collection of mouth-shapes and has to try to form a sentence out of it. The mouth shapes consisted of for parts that can have different forms: upper lip, lower lip, teeth and tongue. One mouth shape can represent one or more letters (or syllables) of the English alphabet. [63]
The detailed description of each task:

- "Level one, two and three were the easier tasks to provide the base for the later levels: Recognize a letter, recognize a word (out of a dictionary), and recognize a word including syllables. Syllables are tricky because they represent two or more letters of the English letters which makes the search for matching words more complex. All possible mouth shapes and which letters/syllables they can represent was provided in the task description and in an additional file. The dictionary held 18609 real English words and was provided in another file inside the task package.

- In level four, the task was to calculate the likelihood that a particular letter or syllable is followed by another letter or syllables (based on the dictionary).
- This provides the syllable likelihood which is used in level five: Since a group of mouth shapes could match many different words, the likelihood for all words has to be calculated and the most likely word will be the result.
- The goal for level six was to calculate the likelihood of the words (interpreted out of mouth forms) inside a sentence. A file with 551 620 words (Collection of Shakespeare texts) provided the base.
- In the final level (level seven) the teams received only a number of mouth shapes and they had to guess the most likely sentence said. Note that this exercise is based on words inside sentences by Shakespeare. " [63]

5.7 Threats to validity

All empirical studies have to consider threats to validity according to Cook & Campbells scheme [17] grouped into conclusion-validity, internal-validity, construct-validity and external validity. In this section limitations and threats are discussed an how they were addressed.

Conclusion validity Team size and experience level of the participants are not influenced by the organizer of the contest. The price money motivates each participant to give his or her best. For the hypothesis testing, T-tests at a 95 % significance level are used.

Internal validity The CCC has a history reaching back to 2007 holding annually contests. Since 2011 there are three CCC per year. The contest setup has been proved several times.
There is the risk, that teams help each other out. So, communication between the participants has not been allowed, except within the teams itself. Internet connectivity was available and mobile phones were not prohibited which enables the possibility that teams communicated online. However, the price money gained by the best teams highly reduces the motivation to help another team because you risk loosing price money to them.
Skill level and team size are asked before the contest but contestants may state wrong information. All information about the participants are believed to be correct. Only obviously wrong user data is filtered out of the results.

Construct validity The experiment is based on related work and previous CCCs and the resulting variables, duration and defect count, are common in empirical studies.
One problem that can arise is that the experiment duration (CCCs last from two to four hours) is too short to meaningfully test software methodologies (like Pair-Programming in RI3). However, this is a fixed variable that cannot be changed by the researcher. It is expected that participants that attend in pairs already know each other and performed Pair-Programming in previous situations together so that no settling-in period is needed.

External validity The high number of participants (around 300 programmers, forming around 150 teams) with different backgrounds (Juniors, Intermediates, Professionals) that are attending the CCCs, provide a representative sample of the software engineers population. A classroom setting was used to monitor and control study variables.
The teams are formed by choice of the contestants. This can reduce the certainty that teams are as experienced as single programmers. Better participants may choose to work alone while juniors may prefer to work in a team.

CHAPTER 6

Results

According to Wohlin [64] the result chapter is split into three sections: Descriptive statistics, data set reduction and hypotheses testing.

6.1 Descriptive statistics

In this section the result data of the experiment will be presented using descriptive statistics methods. Like described in chapter 5, the result of the data was extracted out of Excel sheets or SQL database tables and moved into an IBM SPSS Statistic 20 database. SPSS was used to create the following tables, plots and graphics.

6.1.1 General statistics

In this subsection a general overview in numbers is presented. Table 6.1 shows the total numbers of participating teams, however this can also be single programmers but from now on they are referred to as 'team'. In total 173 teams finished at least level one and the programming language that every team used is known. From 173 teams 26 have an invalidated team size, i.e. it is not known if they participated alone, in a pair or a group of three. Further from the 173 teams one team had an invalidated experience level so that it cannot be grouped into junior, intermediate and professional.

Statistics

		Completed Levels	Coding language	Team Size	Experience Level
N	Valid	173	173	147	172
	Missing	0	0	26	1

Table 6.1: Shows the number of data sets obtained during the experience and how many are valid and missing.

Completed Levels

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	44	25,4	25,4	25,4
	2	81	46,8	46,8	72,3
	3	28	16,2	16,2	88,4
	4	6	3,5	3,5	91,9
	5	1	,6	,6	92,5
	6	7	4,0	4,0	96,5
	7	6	3,5	3,5	100,0
	Total	173	100,0	100,0	

Table 6.2: Shows which level was completed by how many teams. (Only counting the highest completed level per team.)

Location

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Vienna	119	68,8	68,8	68,8
	Cluj	54	31,2	31,2	100,0
	Total	173	100,0	100,0	

Table 6.3: Shows where the participating teams were located.

Source code available

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	No	28	16,2	16,2	16,2
	Yes	145	83,8	83,8	100,0
	Total	173	100,0	100,0	

Table 6.4: Shows how many teams uploaded their source code for one or more completed levels.

Table 6.2 shows the progress of teams grouped by level. I.e. each row shows a level and the number of teams that progressed until this level. The first shows that 44 teams, out of 173, (only) completed level one. 81 teams reached level two during the contest, etc. Only six teams solved all seven levels.
Conclusively, the majority of the teams stuck in the levels 1-3 (88.4 %), thus the hypotheses testing should focus on these levels to maximize its validity.

Table 6.3 is a statistic about the two locations where the contest was held simultaneously. 119 teams competed in Vienna while 54 teams participated in Cluj, i.e. 68.8 % of the teams were in Vienna.

Table 6.4 summarizes how many teams uploaded their source code during the contest. Since teams got a time bonus when uploading their code, most teams made use of this offer and uploaded the code (83.8 %).

Coding language

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	C/C++	55	31,8	31,8	31,8
	Java	44	25,4	25,4	57,2
	C#	36	20,8	20,8	78,0
	Python	12	6,9	6,9	85,0
	PHP	6	3,5	3,5	88,4
	Ruby	4	2,3	2,3	90,8
	Groovy	2	1,2	1,2	91,9
	Haskell	2	1,2	1,2	93,1
	JavaScript	2	1,2	1,2	94,2
	Scala	2	1,2	1,2	95,4
	Mathlab	2	1,2	1,2	96,5
	Other	2	1,2	1,2	97,7
	Objective C	1	,6	,6	98,3
	Perl	1	,6	,6	98,8
	Smalltalk	1	,6	,6	99,4
	LUA	1	,6	,6	100,0
	Total	173	100,0	100,0	

Table 6.5: Usage distribution of programming languages during the experiment

Table 6.5 shows the used programming languages by all teams. C/C++ was the most used programming language with 55 teams (or 31.8 %), followed by Java with 44 teams (25.4 %) and C# with 36 teams (20.8 %). Together, 78 % of all participants used one of these languages.
C and C++ had been combined to one group since they use the same compiler and many teams used their functionalities in mixed forms.

Table 6.6 gives an overview of the team size distribution. Unfortunately, 26 out of 173 teams have an invalidated team size. For various reasons there is no information in the result data about these teams. However, their data sets will only be filtered out if it is required by the hypotheses testing.
The majority of the teams worked alone, namely 81 teams out of 147 (or 55.1 %). 52 teams used Pair-Programming (35.4 %) and 14 teams worked in a team of three (9.5 %).

Table 6.7 shows the experience distribution of the participating teams. Because of incomplete data, one data set has a missing value so that only 172 teams can be rated according to their experience.
The largest group were the intermediates (mainly college students) with 93 teams (or 54.1 %). Followed by the nearly equal large groups of juniors (40 teams or 23.3 %) and professionals (39 teams or 22.7 %).

The last table of this subsection 6.8 shows a cross tabulation of the previous two tables about team size and experience level and how many levels each combination was able to complete. This tabulation is important because when comparing the results of each group to each other,

Team Size

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	81	46,8	55,1	55,1
	2	52	30,1	35,4	90,5
	3	14	8,1	9,5	100,0
	Total	147	85,0	100,0	
Missing		26	15,0		
Total		173	100,0		

Table 6.6: Shows the distribution of teams regarding their size.

Experience Level

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Junior	40	23,1	23,3	23,3
	Intermediate	93	53,8	54,1	77,3
	Professional	39	22,5	22,7	100,0
	Total	172	99,4	100,0	
Missing		1	,6		
Total		173	100,0		

Table 6.7: Shows the distribution of teams regarding their experience levels.

only one factor shall be evaluated at a time.
Thus, RI3 will compare junior individuals against junior pairs, intermediate individuals against intermediate pairs and expert individuals against professionals pairs.
According to the same logic, junior, intermediate and professional individuals will be compared against each other, as well as junior, intermediate and professional pairs with each other.

6.1.2 Success in the competition

This subsection will present the results from the competition and show how team size and experience groups competed against each other. Solving levels is the key to be successful in the Catalysts Coding Contest because the final ranking is compounded by the number of levels solved, duration and defects (in this order). Each defect is penalized by an increase of one minute on duration.

Figure 6.1 shows how individuals, pairs and teams of three competed regarding the number of levels solved. The median of all three groups is two but individuals had a wider distribution to the top so that the lower quartile lays at two and the higher quartile at three. The lower and higher quartile of pairs and teams of three, by contrast, lays at one and two. This means that individuals performed better, i.e. achieved more levels, than pairs and teams of three. Notice, that the minimum is one, since groups with no completed levels were removed in the validation phase.
Individuals and pairs both show mild and extreme outliers for the higher levels. On the other

Cross tabulation: Team Size - Experience Level - Completed levels

			Experience Level			
Completed Levels			Junior	Intermediate	Professional	Total
1	Team Size	1	5	12	3	20
		2	8	6	2	16
		3	3	.	.	3
2	Team Size	1	4	18	10	32
		2	5	12	8	25
		3	3	4	.	7
3	Team Size	1	1	9	6	16
		2	2	.	2	4
		3	.	2	1	3
4	Team Size	1	1	2	1	4
		2	1	1	.	2
		3	.	.	.	.
5	Team Size	1	.	1	.	1
		2	.	.	.	.
		3	.	.	.	.
6	Team Size	1	.	2	2	4
		2	1	1	1	3
		3	.	.	.	.
7	Team Size	1	.	4	.	4
		2	.	2	.	2
		3	.	.	.	.
Total	Team Size	1	11	48	22	81
		2	17	22	13	52
		3	6	6	1	13
		Total	34	76	36	146

Table 6.8: Shows the cross tabulation of team size and experience level and how many levels each group completed. The leftmost column tells the maximum number of completed levels. The next two columns filter the team size (individuals, pairs and teams of three). Following, three columns for the experience levels and one column showing the total numbers for each level. The last rows show the total numbers for all levels combined.

hand, pairs of three don't show outliers. The numbers written next to the outliers show up to four ranks, reached by participants of these groups.
The quartiles and outliers show what was already mentioned in the description of table 6.2: Level four and higher was only reached by a minority, while most teams only solved 1-3 levels.

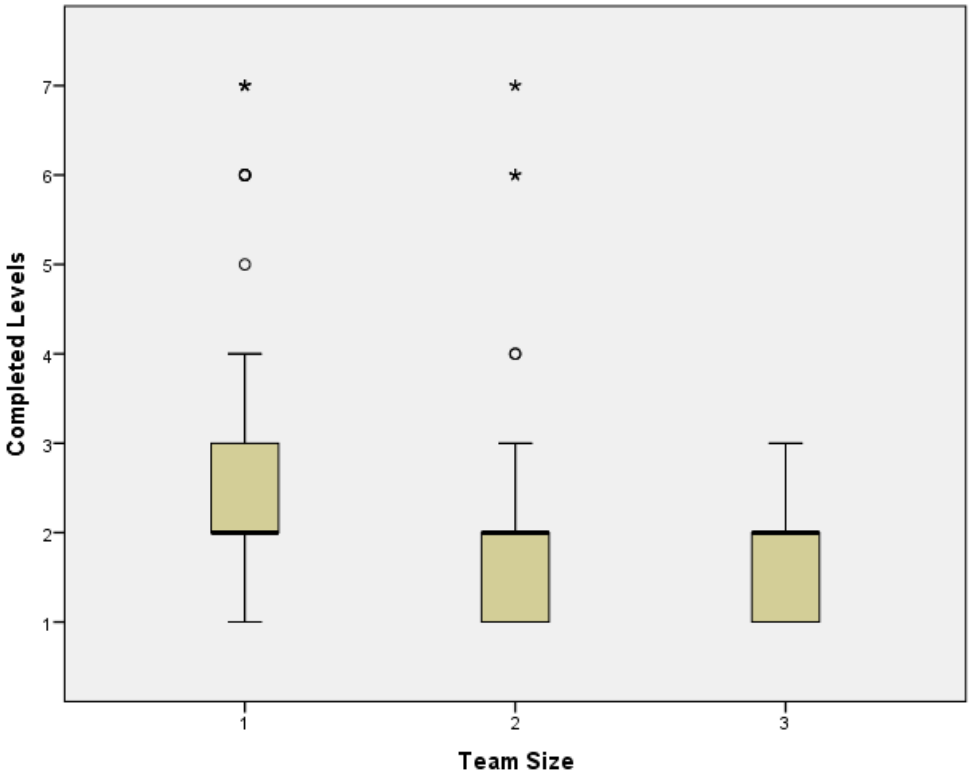


Figure 6.1: This boxplot shows the performance of groups with different team sizes regarding their number of completed levels. The numbers around the outliers describe the total rank, that was reached by one group.

Figure 6.2 is a boxplot that shows a comparison of the experience groups regarding levels solved. The median of all three groups lays at two. Juniors have a 25 quartile of one and 75 quartile of two, intermediates and professionals reach from two to three. This shows, that juniors had a weaker performance than intermediates and professionals.
All three groups show outliers to the top, especially the intermediates, who were the only group that reached the final level seven. The numbers written next to the outliers show up to four ranks, reached by participants of these experience groups.
Alike figure 6.1, figure 6.2 suggests, that any research on profession and team size should focus on the groups that solved level 1-3 only.

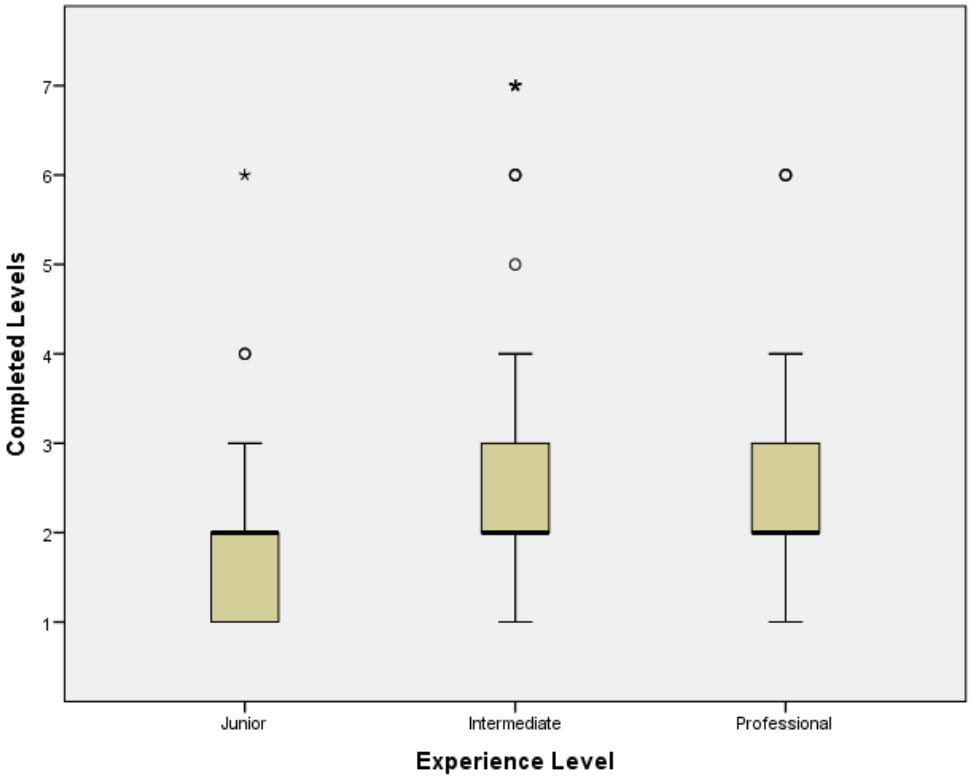


Figure 6.2: This boxplot shows the performance of groups with different experience regarding their number of completed levels. The numbers around the outliers describe the total rank, that was reached by one group.

6.1.3 RI3: Duration

In this subsection the results needed to test the first hypothesis of research issue 3 are presented. Hypothesis 3.1 checks the duration needed by individuals and teams to solve a level (task) of the contest. The alternative hypothesis suggests that teams work faster than individuals.

Table 6.9 and 6.10 show the mean duration, the amount of data rows and the standard deviation for all three group sizes (individuals, pairs, teams of three).
Since the table was too wide to contain the data of all seven levels at once it was split into two tables. Table 6.9 represents the levels 1-3 and table 6.10 the levels 4-7.

Level 1-3 seem to be the more elaborated levels with an average duration needed to complete it of around 60 minutes each. Levels 4 and 7 were completed in 45 minutes on average and 5-6 in 17-20 minutes.

When comparing pairs to individuals, the individuals were faster on average in level two and three. In the other five levels (1, 4, 5, 6, 7) the pairs were faster. The differences amounted between two and twelve minutes.
Expressing the results using percentage scale, pairs had the following speed increases(negative

numbers) or decreases(positive numbers) compared to individuals (Levels 1-7): -5.50%, 16.80%, 20.87%, -5.34%, -10.86%, -13.23% and -20.96%. So pairs were 5.5% faster in level 1. etc.

Teams of three were the weakest teams in all levels. In level 1-3 it took them up to 40 minutes longer on average to complete a level, while none of them completed level 4-7 at all.

Team size and Duration: Levels 1-3

Team Size		Level 1	Level 2	Level 3
1	Mean	0:58:29	1:02:00	0:55:50
	N	81	61	29
	Std. Deviation	0:37:47	0:37:17	0:35:57
2	Mean	0:55:16	1:12:25	1:07:29
	N	52	36	11
	Std. Deviation	0:31:39	0:42:42	0:37:21
3	Mean	1:17:53	1:18:52	1:40:05
	N	14	10	3
	Std. Deviation	1:00:24	0:44:12	0:55:19
Total	Mean	0:59:12	1:07:05	1:01:54
	N	147	107	43
	Std. Deviation	0:38:41	0:39:54	0:38:26

Table 6.9: Shows the duration needed by the participants to complete level 1-3, grouped by team size: individuals, pairs and teams of three.

Team size and Duration: Levels 4-7

Team Size		Level 4	Level 5	Level 6	Level 7
1	Mean	0:46:30	0:17:39	0:20:47	0:48:50
	N	13	9	8	4
	Std. Deviation	0:23:49	0:09:31	0:16:07	0:23:45
2	Mean	0:44:01	0:15:44	0:18:02	0:38:36
	N	7	5	5	2
	Std. Deviation	0:25:30	0:06:57	0:08:37	0:15:35
3	Mean				
	N	0	0	0	0
	Std. Deviation				
Total	Mean	0:45:38	0:16:58	0:19:43	0:45:25
	N	20	14	13	6
	Std. Deviation	0:23:46	0:08:27	0:13:21	0:20:22

Table 6.10: Shows the duration needed by the participants to complete level 4-7, grouped by team size: individuals, pairs and teams of three.

6.1.4 RI3: Effort

In this subsection the results needed for hypothesis 3.2 are presented, namely the effort needed by individuals, pairs and teams of three to finish a level. The effort presents the 'man-hours' needed for a task to be completed, so the duration of pairs was multiplied by two and the duration of teams of three was multiplied by three.

Table 6.11 and 6.12 show the mean effort, the amount of data rows and the standard deviation for all three group sizes (individuals, pairs, teams of three).
The tables were split to fit into the page width. Table 6.11 represents the levels 1-3 and table 6.12 the levels 4-7.

When comparing the effort of individuals against the effort of pairs, the individuals have clearly better results. Since the duration statistics above didn't show a 50% or more percent speed increase for pairs, individuals automatically needed less effort to complete the tasks.
The percentages of effort increase for level 1-7 when comparing Pair-Programming to individuals are: 89.00%, 133.60%, 141.73%, 89.32%, 78.28%, 73.54% and 58.09%. This means that for level two and three the effort more than doubled.

Team size and Effort: Levels 1-3

Team Size		Level 1	Level 2	Level 3
1	Mean	0:58:29	1:02:00	0:55:50
	N	81	61	29
	Std. Deviation	0:37:47	0:37:17	0:35:57
2	Mean	1:50:33	2:24:51	2:14:58
	N	52	36	11
	Std. Deviation	1:03:18	1:25:24	1:14:43
3	Mean	3:53:41	3:56:36	5:00:15
	N	14	10	3
	Std. Deviation	3:01:13	2:12:37	2:45:57
Total	Mean	1:33:35	1:46:12	1:33:07
	N	147	107	43
	Std. Deviation	1:28:10	1:29:01	1:29:21

Table 6.11: Shows the effort needed by the participants to complete level 1-3, grouped by team size: individuals, pairs and teams of three.

6.1.5 RI3: Defects

In this subsection the results needed for hypothesis 3.3 is presented, namely the defects made by individuals, pairs and teams of three in each level. A defect is a failed acceptance test case in the CCC. Only when all acceptance tests are successful the participant can move to the next level. The alternative hypothesis 2.3 state that pairs will make fewer mistakes than individuals.

Table 6.13 and 6.14 show the mean defects, the amount of data rows and the standard deviation per level for all three group sizes (individuals, pairs, teams of three).

Team size and Effort: Levels 4-7

Team Size		Level 4	Level 5	Level 6	Level 7
1	Mean	0:46:30	0:17:39	0:20:47	0:48:50
	N	13	9	8	4
	Std. Deviation	0:23:49	0:09:31	0:16:07	0:23:45
2	Mean	1:28:02	0:31:29	0:36:04	1:17:13
	N	7	5	5	2
	Std. Deviation	0:51:01	0:13:55	0:17:15	0:31:11
3	Mean				
	N	0	0	0	0
	Std. Deviation				
Total	Mean	1:01:02	0:22:35	0:26:40	0:58:17
	N	20	14	13	6
	Std. Deviation	0:39:55	0:12:45	0:17:38	0:27:20

Table 6.12: Shows the effort needed by the participants to complete level 4-7, grouped by team size: individuals, pairs and teams of three.

The tables were split to fit into the page width. Table 6.13 represents the levels 1-3 and table 6.14 the levels 4-7.

With one exception (level 6) pairs made on average fewer defects than individuals. However, because of the lower sample size in the higher levels (4-7) these levels are not as meaningful as level 1-3.
Expressed in percentages the decreases (negative values) and increases (positive values) of defects made when comparing pairs to individuals (levels 1-7): -26.57%, -9.43%, -13.50%, -31.51%, -91.42%, 361.54% and -71.43%. In words, pairs made 26.57% fewer defects in level one than individuals.

Teams of three, by contrast, made more defects in level one and three than the other two groups, but made the fewest in level two.

6.2 Data set reduction

According to Wohlin et al. [64] the results will be checked for outliers. Outliers can lead to wrong descriptive analysis and harm the validity of t-tests for hypothesis testing because if one group contained outliers and the comparing group didn't, the mean and deviation values are distorted.

Figure 6.3 shows a scatter plot for level ones duration and defects. Wohlin et al. [64] suggests scatter plots identify atypical data points. Due the high density close to zero defects and zero to two hours duration there seem to be extreme outliers on both axes. The presence of the defect outliers at 42 and 72 raises the mean value for these groups drastically. The scatter plot illustrates, that there are extreme outliers in our data set which can affect the expressiveness of our ANOVA tests later.

Team size and Defects: Levels 1-3

Team Size		Level 1	Level 2	Level 3
1	Mean	2,86	1,59	2,00
	N	81	61	29
	Std. Deviation	8,793	2,597	2,686
2	Mean	2,10	1,44	1,73
	N	52	36	11
	Std. Deviation	5,112	2,049	3,003
3	Mean	4,07	1,10	6,67
	N	14	10	3
	Std. Deviation	5,929	1,524	3,786
Total	Mean	2,71	1,50	2,26
	N	147	107	43
	Std. Deviation	7,413	2,329	3,024

Table 6.13: Shows the defects made by the participants while solving level 1-3, grouped by team size: individuals, pairs and teams of three.

Team size and Defects: Levels 4-7

Team Size		Level 4	Level 5	Level 6	Level 7
1	Mean	1,46	2,33	,13	1,75
	N	13	9	8	4
	Std. Deviation	2,145	3,354	,354	2,062
2	Mean	1,00	,20	,60	,50
	N	7	5	5	2
	Std. Deviation	2,236	,447	,894	,707
3	Mean				
	N	0	0	0	0
	Std. Deviation				
Total	Mean	1,30	1,57	,31	1,33
	N	20	14	13	6
	Std. Deviation	2,130	2,848	,630	1,751

Table 6.14: Shows the defects made by the participants while solving level 4-7, grouped by team size: individuals, pairs and teams of three.

Thus, figure 6.4 shows the same values like in figure 6.3 but as boxplot instead of scatterplot. Thus, the mild and extreme outliers are made visual. On the left half of figure 6.4, the duration boxplot is presented. There are five mild and two extreme outliers. The numbers next to the outliers represent the total ranking of these teams.
The right half shows the defects-boxplot and the extreme outliers with 15 and more defects per team. This high number of defects can be explained by the use of brute-force-like attempts to

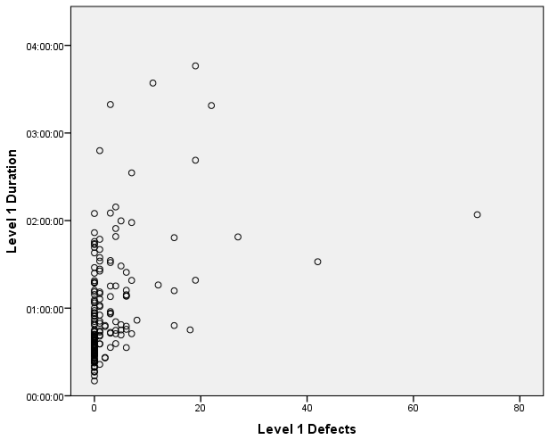


Figure 6.3: A scatterplot showing the duration and defects made per each participating team in level one. Defects cluster at zero and duration clusters at around one hour. This figure illustrates that the result data is scattered, which can affect the expressiveness of the mean value and variance tests (ANOVA) in a bad way.

solve a level or major misunderstandings in the task descriptions.
Wohlin et al. [64] mention, that it should be well thought through if outliers shall be removed from the data set: If it is likely that an event that led to an outlier will occur again then it shouldn't be removed since these outliers hold valuable information. On the other hand, the extreme outliers distort the mean values and standard deviations.

Figure 6.5 shows the duration and defects boxplot for level two. There are no outliers for the duration but, like in level one, there are mild and extreme outliers. The extreme outliers were removed.

Figure 6.6 shows the duration and defects boxplot for level three. The duration boxplot shows no outliers but there is one (extreme) outlier in the defects boxplot, which was be removed for further analysis.

However, data points should not be removed based on different limits per level. Considering defects, extreme outliers for level one start at 15 defects, while in level two the extreme outliers start already at eight defects. On the other hand, in level six the number of defects made was extremely low so that every team with one or more defects was treated as an outlier. To provide a balanced data set for the evaluation, a limit of ten defects was defined. A team with more than ten defects in a level will be excluded from this levels statistic.
Regarding duration, in all seven levels there are only two extreme outliers (both in level one). These two data points will be excluded from level one.

In level one, 13 data points were removed: eleven due too many defects and two because of too many defects and too long duration. In level two, the number of removed data points belongs amounts: three and in level three: two.

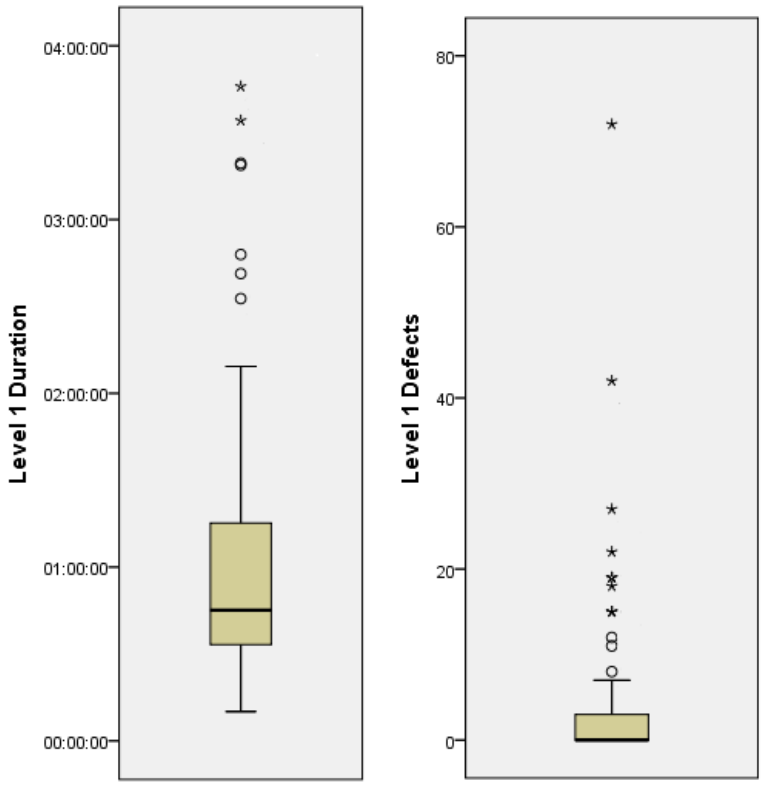


Figure 6.4: Boxplots showing the duration needed to complete level one by the teams on the left side and the defects made in level one on the right side. Both plots contain mild and extreme outliers who are considered to be excluded from the analysis.

6.3 Hypotheses testing

In this section the filtered data sets will be presented and t-tests are run to test the hypotheses. The structure follows the one of the previous sections: First the hypotheses for RI3 will be answered: 3.1, 3.2 and 3.3, afterwards for RI4: 4.1 and 4.2.

For each subsection, the general results and its t-test are presented first. Then the data is grouped into experience levels (for RI3) or team sizes (for RI4) and the results of each subgroup and its corresponding t-test are given.
This grouping is necessary to make sure that only one variable/factor is analyzed.

6.3.1 RI3: Duration

In this subsection, the data for hypothesis 3.1 will be summarized and tested. The hypothesis said that pairs are faster than individuals.

Like described in section 6.2, 18 data points of level 1-3 have been removed because they were extreme outliers that would harm the validity of the test because one or more groups/subgroups

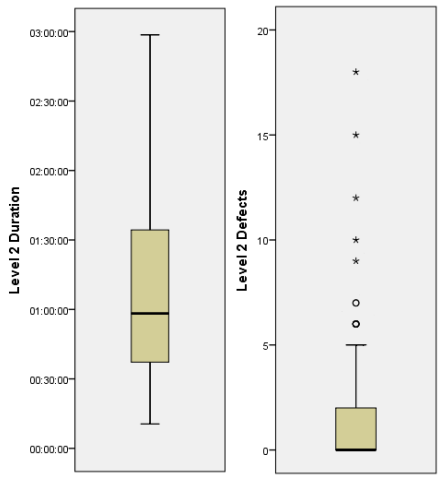


Figure 6.5: Boxplots showing the duration and defects of level two. There are many mild and extreme outliers in the defects boxplot. (right)

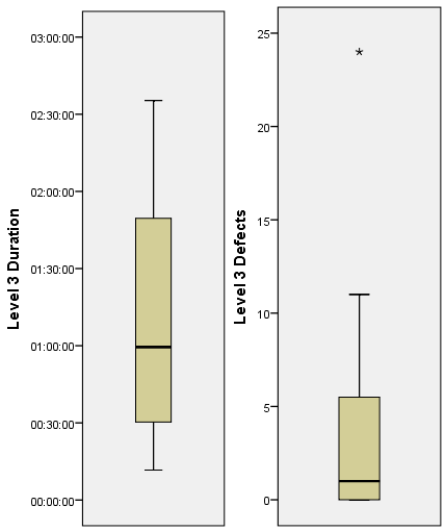


Figure 6.6: Boxplots showing the duration and defects of level three. There is only one (extreme) outlier in the defects boxplot on the right.

could have distorted mean values while comparing groups wouldn't.

Table 6.15 shows the overall level statistic for the duration factor, grouped by team size. Teams of three and teams with unknown team sizes are not presented.

Notice, that there is no filtering regarding experience level. That means that if the experience levels wouldn't be spread equally, it could happen that all individuals are professionals and all pairs are juniors and intermediates. This would lead to a distorted statistic. To address this problem, see table 6.17 to 6.22 where the results are also grouped into experience levels.

Duration: Individuals vs. Pairs (All experiences)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1	1	76	0:52:48	0:28:26	0:03:15
	2	48	0:51:10	0:27:28	0:03:57
Level 2	1	60	1:01:12	0:37:04	0:04:47
	2	36	1:12:25	0:42:42	0:07:07
Level 3	1	29	0:55:50	0:35:57	0:06:40
	2	11	1:07:29	0:37:21	0:11:15
Level 4	1	13	0:46:30	0:23:49	0:06:36
	2	7	0:44:01	0:25:30	0:09:38
Level 5	1	9	0:17:39	0:09:31	0:03:10
	2	5	0:15:44	0:06:57	0:03:06
Level 6	1	8	0:20:47	0:16:07	0:05:42
	2	5	0:18:02	0:08:37	0:03:51
Level 7	1	4	0:48:50	0:23:45	0:11:52
	2	2	0:38:36	0:15:35	0:11:01

Table 6.15: Shows the overall (all experience groups together) level statistic for two groups of interest (pairs and individuals) regarding duration. Each line shows the results for the particular group in a level: the size of the group, the mean duration, the standard deviation and the standard error mean.

The table (6.15) shows the duration that was needed by all participants to complete each of the seven levels. Each level is split into two rows, one for individual programmers and the second line for pairs. Column 'N' shows the number of data points in this group, followed by the columns 'Mean', 'Standard Deviation' and 'Standard Error Mean'.

Across all levels, there are more data points of individuals than pairs. In five of the seven levels, teams were faster than individuals, although in the two levels where individuals were faster (level two and three) the difference was higher. Only in level seven, the pairs were much faster than individuals, but the number of participants in this level was very low. The standard deviation in level one to four is high (greater 23 minutes) for both groups. Moreover, the relatively high standard error mean values signal, that the means could still vary: In extreme cases the mean duration of individuals could be lower than the one of pairs, as well as the other way around.

Table 6.16 shows the t-test performed on the data of table 6.15. The t-test is structured into the seven levels. For each level a Levene test evaluated the variances of both groups (individuals and pairs) and suggests if an equal variance is assumed or not. If the significance of the Levene test is below or equal to 0.05, than an equal variance is not assumed and the second row of each level has to take into account. Otherwise (Levene Sig. greater than 0.05), an equal variance is assumed and the first row has to be looked at.

At this table (6.16) an equal variance can be assumed and only the first row for each level is important. Other t-tests of this master thesis will have the unnecessary line removed with the

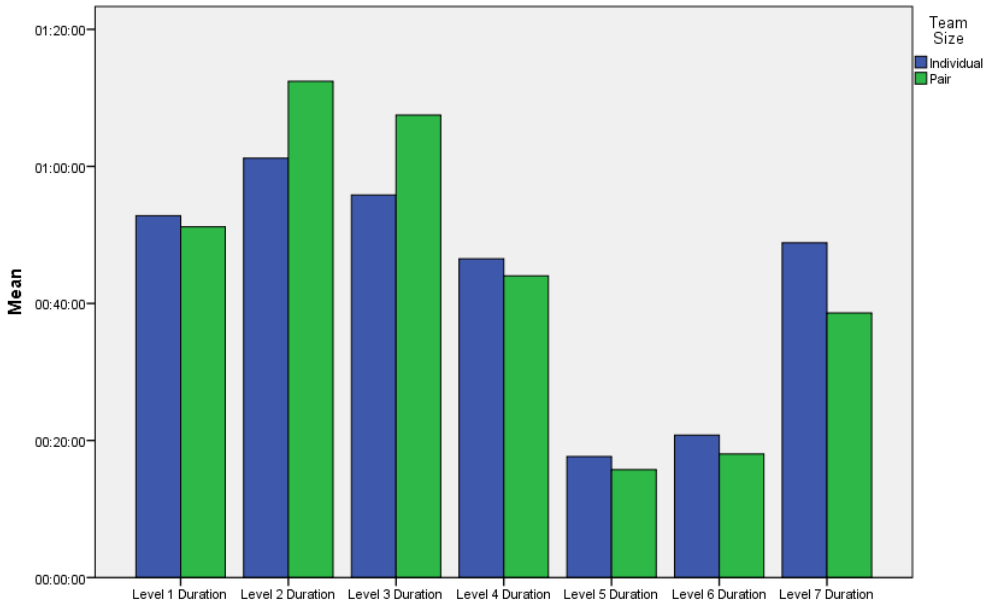


Figure 6.7: Graphical representation of Table 6.15.

information for each particular case which line was chosen. The t-test columns (the last four from the right) show the t-test value, the degrees of freedom, the (two-tailed) significance of the test and the mean difference between the two compared groups. The significance values are all above 0.05, which means that the null hypothesis 3.1.0 cannot be rejected: Pairs and individuals complete tasks equally fast.

Like described in the beginning of this section, the results are now split into experience levels before comparing individuals against pairs.

Table 6.18 shows how juniors competed in the contest regarding duration. In the overall population there have been more individual teams than pairs, but juniors participated more times in pairs than as individuals.

The results for juniors are mixed: In level one and three individuals where faster but in level two and four the pairs. Not junior team completed level seven and only one pair and no individual complete level five and six. In level three the difference between juniors and pairs was 47 minutes but the sample size is low with only six teams. Level two shows that pairs worked ten minutes faster but the standard error mean is very high (17 and 20 minutes) which means the result is not representative. Level one is the only level with a higher sample size (28). The difference amounts to five minutes, but its below the standard error mean of seven respectively nine minutes.

Table 6.18 represents the t-test made on the data of table 6.17. There was no Levene test for level four because there was only one individual participant which means that there is no variation that can be analyzed. For the first three levels equal variances are assumed. The significance of the t-tests is above 0.05 which means the null hypothesis 3.1.0 cannot be rejected for juniors. Although, the significance at level three amounts 0.057 and thus barely missed the

T-Test - Duration: Individuals vs. Pairs (All experiences)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	**assumed**	,711	,401	,313	122,000	,755	0:01:37
	not assumed			,316	102,657	,753	0:01:37
2	**assumed**	,675	,413	-1,357	94,000	,178	-0:11:13
	not assumed			-1,310	65,821	,195	-0:11:13
3	**assumed**	,104	,749	-,905	38,000	,371	-0:11:38
	not assumed			-,889	17,493	,386	-0:11:38
4	**assumed**	,019	,892	,218	18,000	,830	0:02:29
	not assumed			,213	11,672	,835	0:02:29
5	**assumed**	2,178	,166	,390	12,000	,703	0:01:54
	not assumed			,428	10,805	,677	0:01:54
6	**assumed**	,641	,440	,347	11,000	,735	0:02:44
	not assumed			,399	10,884	,697	0:02:44
7	**assumed**	,524	,509	,537	4,000	,620	0:10:13
	not assumed			,631	3,222	,570	0:10:13

Table 6.16: Shows the t-test results for the previous table 6.15. For each level a Levene and t-test is made. The result from the Levene test is styled with bold letters and signals if an equal variance is assumed and thus which line of the t-test has to be used. The most interesting column of this table is the second last "Sig." which shows the significance of the t-test. In none of the seven levels a significance of less than 0.05 was observed meaning that the results of pairs and individuals don't differ much enough to show evidence than one group is faster.

limit. This would have meant that individuals are working faster than pairs. Across all levels there have been more individuals than pairs.

Table 6.19 shows the statistic for intermediate experienced participants, grouped into pairs and individuals. In five of the seven levels, the pairs were faster than the individuals, however the standard error means were high so that the mean values still could change.

Table 6.20 shows the t-tests for intermediate participants that were presented in table 6.19. Except level five all other levels' variances are assumed to be equal. There were no significant differences found which means that the null hypotheses 3.1.0 cannot be rejected. Although, the difference of mean duration in level two was "slightly" significant with a value of 0.082.

Table 6.21 shows the duration results for professionals. Across all levels there have been more individual participants than pairs. Also note, that no professional completed level seven. The number of professional teams drops with level three. Only level one and two have at least 30 data rows. In four of the six levels, pairs have been faster but in level two only marginally and in the other ones it can still be by chance because the standard error mean is high enough so that the results could change and switch position.

The t-test made for table 6.21 in table 6.22 shows, that no significant differences in the

Duration: Individuals vs. Pairs (Juniors)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Duration	1	10	00:54:58	00:20:22	00:06:26
	2	15	00:57:05	00:29:52	00:07:42
Level 2 Duration	1	6	01:28:09	00:50:38	00:20:40
	2	9	01:18:37	00:51:34	00:17:11
Level 3 Duration	1	2	00:39:08	00:18:11	00:12:52
	2	4	01:26:57	00:21:39	00:10:49
Level 4 Duration	1	1	01:11:26		
	2	2	00:41:45	00:35:19	00:24:58
Level 5 Duration	1	0			
	2	1	00:08:19		
Level 6 Duration	1	0			
	2	1	00:10:48		
Level 7 Duration	1	0			
	2	0			

Table 6.17: Shows the level statistic for junior-level pairs and individuals regarding duration. Each line shows the results for the particular group in a level: the size of the group, the mean duration, the standard deviation and the standard error mean.

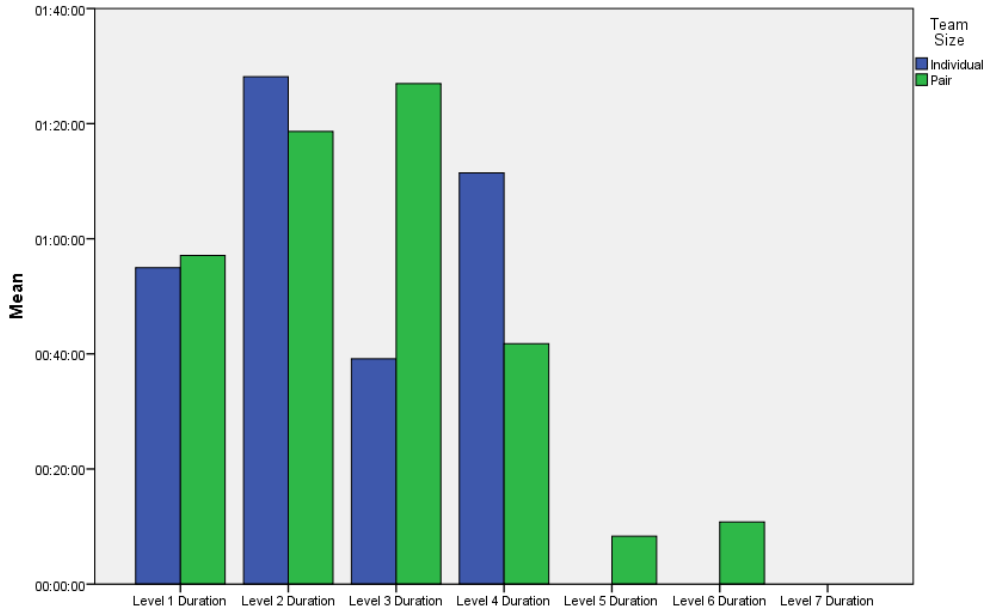


Figure 6.8: Graphical representation of Table 6.17.

T-Test - Duration: Individuals vs. Pairs (Juniors)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	1,721	,202	-,196	23,000	,846	-0:02:07
2	assumed	,065	,803	,353	13,000	,730	0:09:31
3	assumed	,087	,782	-2,649	4,000	,057	-0:47:49
4	assumed	.	.	,686	1,000	,617	0:29:40

Table 6.18: The t-test results of the comparison between junior-level individuals and pairs regarding duration. See table 6.17 for the descriptive data. In level 3 a slight significance was observed (0.057) favoring junior-individuals.

Duration: Individuals vs. Pairs (Intermediates)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Duration	1	44	00:50:23	00:29:40	00:04:28
	2	20	00:50:25	00:26:32	00:05:56
Level 2 Duration	1	35	00:53:04	00:31:04	00:05:15
	2	16	01:12:36	00:46:14	00:11:33
Level 3 Duration	1	18	01:05:51	00:38:14	00:09:01
	2	4	01:11:42	00:49:06	00:24:33
Level 4 Duration	1	9	00:41:45	00:24:48	00:08:16
	2	4	00:33:30	00:03:41	00:01:50
Level 5 Duration	1	7	00:18:13	00:10:01	00:03:47
	2	3	00:14:52	00:04:10	00:02:24
Level 6 Duration	1	6	00:13:54	00:07:32	00:03:05
	2	3	00:17:51	00:09:36	00:05:32
Level 7 Duration	1	4	00:48:50	00:23:45	00:11:53
	2	2	00:38:36	00:15:35	00:11:01

Table 6.19: Shows the level statistic for intermediate-experienced pairs and individuals regarding duration. Each line shows the results for the particular group in a level: the size of the group, the mean duration, the standard deviation and the standard error mean.

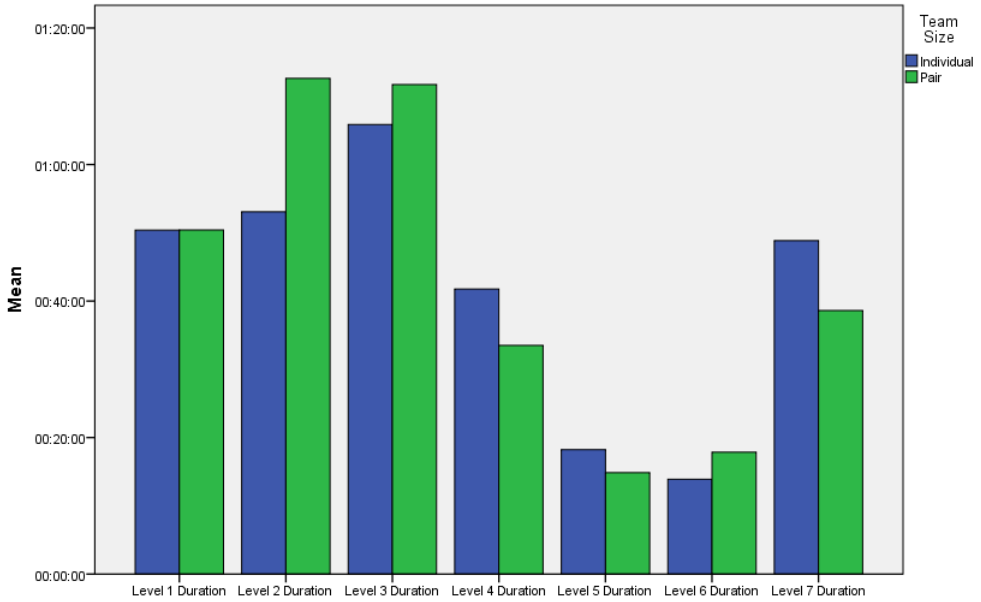


Figure 6.9: Graphical representation of Table 6.19.

T-Test - Duration: Individuals vs. Pairs (Intermediates)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	,604	,440	-,003	62,000	,998	-0:00:01
2	assumed	2,319	,134	-1,778	49,000	,082	-0:19:31
3	assumed	1,072	,313	-,264	20,000	,794	-0:05:50
4	assumed	2,451	,146	,646	11,000	,532	0:08:14
5	not assumed	6,344	,036	,747	7,940	,477	0:03:21
6	assumed	,192	,674	-,684	7,000	,516	-0:03:57
7	assumed	,524	,509	,537	4,000	,620	0:10:13

Table 6.20: The t-test results of the comparison between intermediate-experienced individuals and pairs regarding duration. See table 6.19 for the descriptive data. A slight significance (0.082) was observed in level 2 favoring individuals.

Duration: Individuals vs. Pairs (Professionals)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Duration	1	22	00:56:37	00:29:36	00:06:19
	2	13	00:45:32	00:26:50	00:07:26
Level 2 Duration	1	19	01:07:38	00:39:24	00:09:02
	2	11	01:07:05	00:31:18	00:09:26
Level 3 Duration	1	9	00:39:29	00:27:56	00:09:19
	2	3	00:35:52	00:19:07	00:11:02
Level 4 Duration	1	3	00:52:26	00:21:53	00:12:38
	2	1	01:30:33		
Level 5 Duration	1	2	00:15:39	00:10:35	00:07:30
	2	1	00:25:49		
Level 6 Duration	1	2	00:41:27	00:19:57	00:14:06
	2	1	00:25:49		
Level 7 Duration	1	0			
	2	0			

Table 6.21: Shows the level statistic for professional pairs and individuals regarding duration. Each line shows the results for the particular group in a level: the size of the group, the mean duration, the standard deviation and the standard error mean.

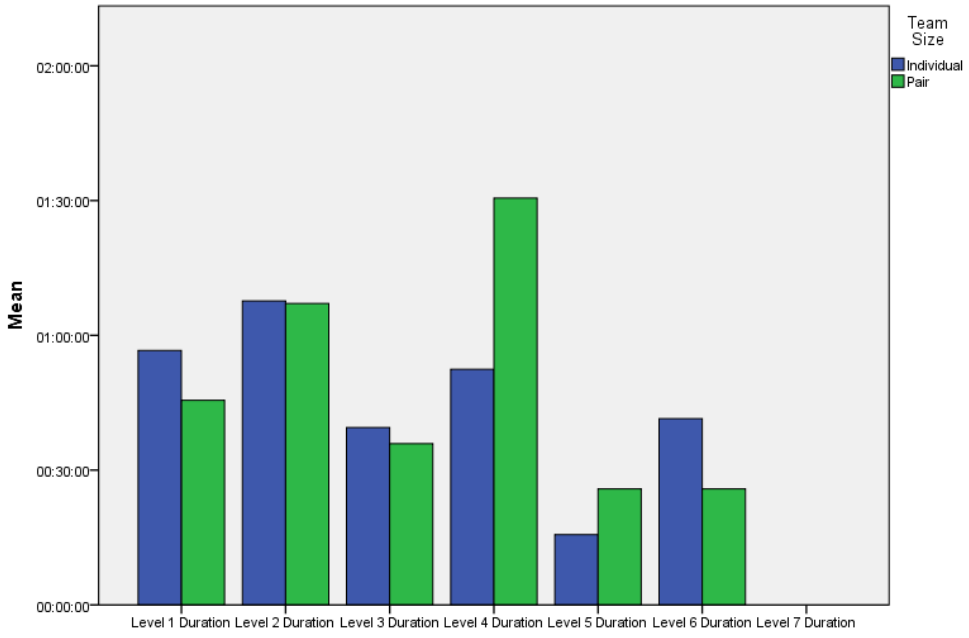


Figure 6.10: Graphical representation of Table 6.21.

T-Test - Duration: Individuals vs. Pairs (Professionals)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	2,511	,123	1,108	33,000	,276	0:11:05
2	assumed	1,115	,300	,039	28,000	,969	0:00:32
3	assumed	,452	,517	,205	10,000	,842	0:03:36
4	assumed	.	.	-1,508	2,000	,271	-0:38:06
5	assumed	.	.	-,783	1,000	,577	-0:10:09
6	assumed	.	.	,640	1,000	,638	0:15:38

Table 6.22: The t-test results of the comparison between professional individuals and pairs regarding duration. See table 6.21 for the descriptive data. No significant differences were observed.

variances between the groups (individuals and pairs) were found.

Filtered by number of achieved levels The participants can also be grouped by the number of levels the completed in total. Table 6.23 shows the results of participants that solved one, two and three levels. Figure 6.11 shows a graphical representation of this table.
Teams that completed more levels are not shown because they were too few in numbers to be expressive. The upper part of the table represents the participants that completed the first level only. The duration of pairs and individuals only differed by 16 seconds.
The middle part represents the participants that completed exactly two levels and it is split into two further parts: their level one results and their level two results. In level one, the pairs were more than ten minutes faster than the individuals, in level two, three and a half minutes slower. The ten-minute-difference proved to be significant as seen in table 6.24.
The lowest part represents the participants that completed the first three levels. Like above, it is split into further groups for each level. In level one, the pairs were 38 seconds faster than individuals, in level two 13 minutes slower and in level three around one minute slower.
Because of removed outliers, the number (N) of data points can vary.

6.3.2 RI3: Effort

In this subsection, the data for hypothesis 3.2 will be summarized and tested. The hypothesis said, that pairs require less effort than individuals.

Table 6.25 shows the overall results of the experiment regarding effort of the teams, grouped by individuals and pairs. Teams of three and teams with unknown team sizes are not presented.

The results show, that pairs need more effort than individuals across all levels. This finding is no surprise, since there were no significant differences in the time needed to complete the tasks (See duration-section 6.3.2). In all seven levels, the effort needed by pairs amounts around double the value needed by individuals.

Duration: Individuals vs. Pairs (Completed Levels)

Number completed levels		Team Size	N	Mean	Std. Deviation	Std. Error Mean
1 Level	Level 1 Duration	1	17	1:24:35	0:25:03	0:06:04
		2	13	1:24:19	0:30:24	0:08:26
2 Levels	Level 1 Duration	1	30	0:51:57	0:22:29	0:04:06
		2	24	0:41:41	0:11:53	0:02:25
	Level 2 Duration	1	31	1:22:55	0:37:44	0:06:46
		2	25	1:26:26	0:41:26	0:08:17
3 Levels	Level 1 Duration	1	16	0:39:17	0:15:46	0:03:56
		2	4	0:38:39	0:06:21	0:03:10
	Level 2 Duration	1	16	0:42:55	0:17:00	0:04:15
		2	4	0:55:39	0:36:01	0:18:00
	Level 3 Duration	1	16	1:00:45	0:37:11	0:09:17
		2	4	1:01:29	0:21:19	0:10:39

Table 6.23: Shows the level statistic for pairs and individuals regarding duration, grouped by the number of levels the completed in total (one to three). Each line shows the results for the particular group in a level: the size of the group, the mean duration, the standard deviation and the standard error mean.

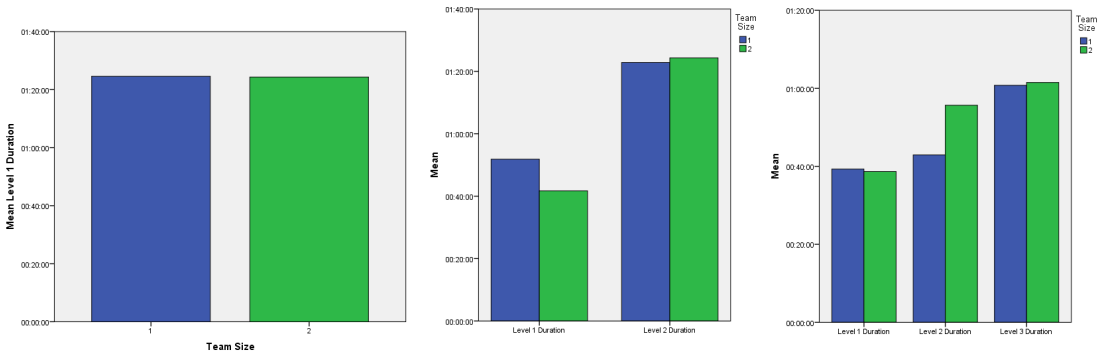


Figure 6.11: Graphical representation of Table 6.23. On the left side, the participants that completed exactly one level, in the middle the teams with two completed levels and on the right side three levels.

T-Test - Duration: Individuals vs. Pairs (Completed Levels)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	,632	,433	,026	28	,980	0:00:15
1	not assumed	9,449	,003	2,151	46	,037	0:10:15
2	assumed	,210	,649	-,332	54	,741	-0:03:31
1	assumed	1,852	,190	,076	18	,940	0:00:37
2	assumed	3,389	,082	-1,065	18	,301	-0:12:44
3	assumed	2,304	,146	-,038	18	,970	-0:00:44

Table 6.24: The t-test results of the comparison between individuals and pairs that completed one, two or three levels, regarding duration. See table 6.23 for the descriptive data. In the second row, pairs completed level one significantly faster than individuals (Sig 0,037).

Effort: Individuals vs. Pairs (All experiences)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1	1	76	0:52:48	0:28:26	0:03:15
	2	48	1:42:21	0:54:57	0:07:55
Level 2	1	60	1:01:12	0:37:04	0:04:47
	2	36	2:24:51	1:25:24	0:14:14
Level 3	1	29	0:55:50	0:35:57	0:06:40
	2	11	2:14:58	1:14:43	0:22:31
Level 4	1	13	0:46:30	0:23:49	0:06:36
	2	7	1:28:02	0:51:01	0:19:17
Level 5	1	9	0:17:39	0:09:31	0:03:10
	2	5	0:31:29	0:13:55	0:06:13
Level 6	1	8	0:20:47	0:16:07	0:05:42
	2	5	0:36:04	0:17:15	0:07:43
Level 7	1	4	0:48:50	0:23:45	0:11:52
	2	2	1:17:13	0:31:11	0:22:03

Table 6.25: Shows the overall (all experience groups together) level statistic for two groups of interest (pairs and individuals) regarding effort. Each line shows the results for the particular group in a level: the size of the group, the mean effort, the standard deviation and the standard error mean.

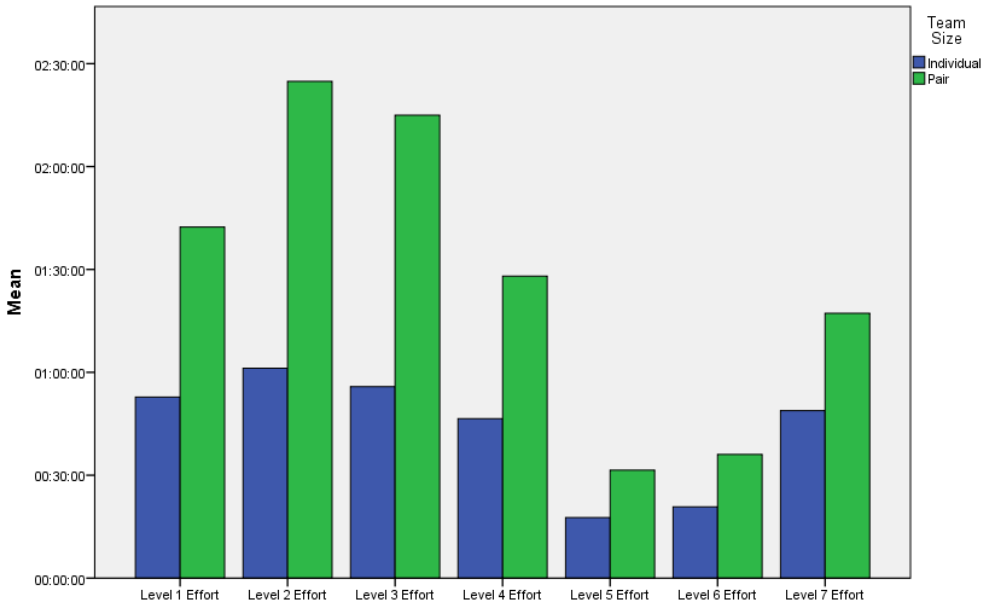


Figure 6.12: Graphical representation of Table 6.25.

Table 6.26 shows the results of the t-test made to search for significant differences in the mean effort values of individuals and pairs. The Levene tests for the first four levels suggested that the variances are not equal. The first three levels show significances of under 0.05 which means that the null hypothesis 3.2.0 can be rejected. However, the alternative hypothesis 3.2.1 cannot be accepted because pairs need more instead of less effort.

Table 6.27 shows the effort results focused on junior participants. Due the low number of junior participants, only the first levels have a sufficient number of data points. No junior passed level seven and no junior-individual completed the levels five or six. The effort of junior-pairs is higher than the effort of junior-individuals across all levels and the standard error mean suggests that this effect is not by chance.

Table 6.28 shows the results of the t-test for junior efforts. The mean values of the first and third level are significantly different. The null hypothesis 3.2.0 can be rejected for juniors but the alternative 3.2.1 cannot be accepted because pairs needed more effort than juniors. Level two showed a significance of 0.11 which means the null cannot be rejected. Level four only had three data rows which means the t-test is not representative and also shows no significant differences.

Table 6.29 shows the results for intermediate individuals and teams. The effort needed by teams was higher across all levels and the standard error mean suggests, that these results are representative.

Table 6.30 shows the results of the t-test for the values of table 6.29. The first two levels have significant differences in the mean values between intermediate-individuals and pairs. The null hypothesis 3.2.0 can be rejected for the first two levels. The alternative hypothesis 3.2.1 cannot be accepted because pairs need more effort than individuals. Level three to seven didn't

T-Test - Effort: Individuals vs. Pairs (All experiences)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	not assumed	13,729	,000	-5,779	63	,000	-0:49:33
2	not assumed	28,531	,000	-5,571	43	,000	-1:23:39
3	not assumed	16,215	,000	-3,368	12	,006	-1:19:08
4	not assumed	5,124	,036	-2,037	7	,079	-0:41:31
5	assumed	1,281	,280	-2,219	12	,046	-0:13:50
6	assumed	,478	,504	-1,621	11	,133	-0:15:17
7	assumed	,231	,656	-1,269	4	,273	-0:28:22

Table 6.26: The t-test results of the comparison between individuals and pairs (of all experience levels) regarding effort. See table 6.25 for the descriptive data. For the first three levels a strongly significant difference (lesser 0.01) was observed, favoring the individuals. In level five the significance was normal (lesser 0.05) and in level four slight (lesser 0.1). Also note, that the Levene test attested not equal variances for the first four levels.

Effort: Individuals vs. Pairs (Juniors)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Effort	1	10	00:54:58	00:20:22	00:06:27
	2	15	01:54:11	00:59:45	00:15:25
Level 2 Effort	1	6	01:28:09	00:50:38	00:20:40
	2	9	02:37:15	01:43:08	00:34:23
Level 3 Effort	1	2	00:39:08	00:18:11	00:12:52
	2	4	02:53:55	00:43:19	00:21:40
Level 4 Effort	1	1	01:11:26		
	2	2	01:23:31	01:10:38	00:49:57
Level 5 Effort	1	0			
	2	1	00:16:38		
Level 6 Effort	1	0			
	2	1	00:21:36		
Level 7 Effort	1	0			
	2	0			

Table 6.27: Shows the level statistic for junior-level pairs and individuals regarding effort. Each line shows the results for the particular group in a level: the size of the group, the mean effort, the standard deviation and the standard error mean.

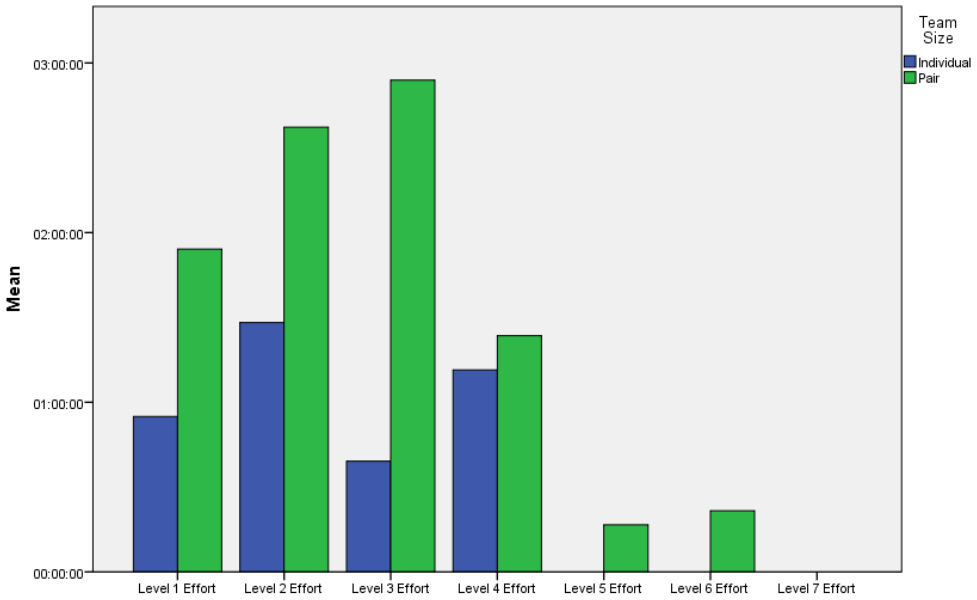


Figure 6.13: Graphical representation of Table 6.27.

T-Test - Effort: Individuals vs. Pairs (Juniors)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	not assumed	8,730	,007	-3,542	18,437	,002	-0:59:13
2	not assumed	5,740	,032	-1,723	12,266	,110	-1:09:06
3	assumed	1,006	,373	-4,032	4,000	,016	-2:14:47
4	assumed	.	.	-,140	1,000	,912	-0:12:05

Table 6.28: The t-test results of the comparison between junior-level individuals and pairs regarding effort. See table 6.27 for the descriptive data. Level one and three show a significant difference between individual juniors and pairing juniors, favoring the individuals.

Effort: Individuals vs. Pairs (Intermediates)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Effort	1	44	00:50:23	00:29:41	00:04:28
	2	20	01:40:50	00:53:05	00:11:52
Level 2 Effort	1	35	00:53:04	00:31:04	00:05:15
	2	16	02:25:13	01:32:30	00:23:07
Level 3 Effort	1	18	01:05:51	00:38:15	00:09:01
	2	4	02:23:25	01:38:13	00:49:06
Level 4 Effort	1	9	00:41:45	00:24:49	00:08:16
	2	4	01:07:01	00:07:23	00:03:42
Level 5 Effort	1	7	00:18:13	00:10:02	00:03:47
	2	3	00:29:44	00:08:21	00:04:49
Level 6 Effort	1	6	00:13:54	00:07:32	00:03:05
	2	3	00:35:43	00:19:13	00:11:06
Level 7 Effort	1	4	00:48:50	00:23:46	00:11:53
	2	2	01:17:13	00:31:11	00:22:03

Table 6.29: Shows the level statistic for intermediate-experienced pairs and individuals regarding effort. Each line shows the results for the particular group in a level: the size of the group, the mean effort, the standard deviation and the standard error mean.

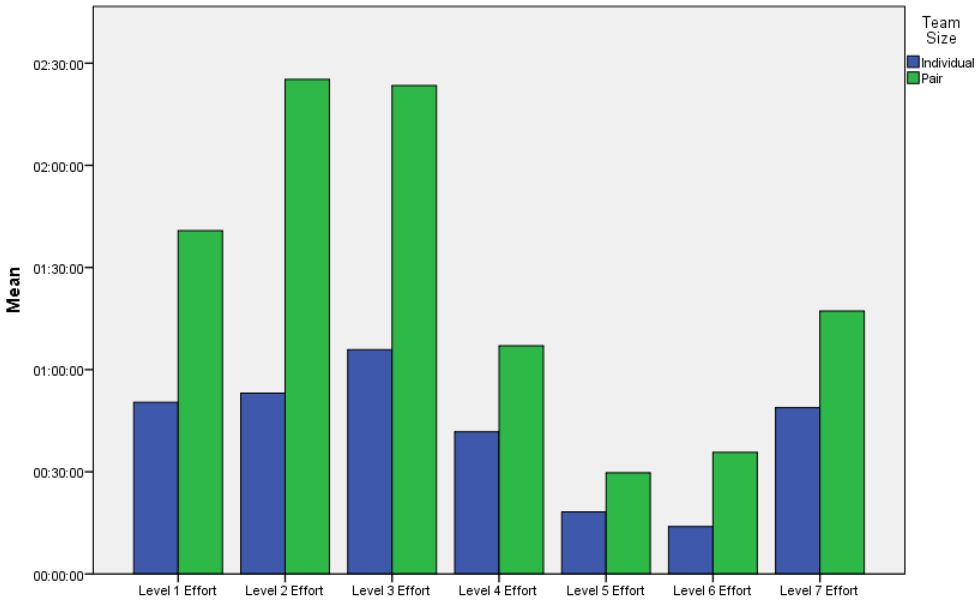


Figure 6.14: Graphical representation of Table 6.29.

T-Test - Effort: Individuals vs. Pairs (Intermediates)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	not assumed	5,281	,025	-3,976	24,564	,001	-0:50:26
2	not assumed	17,897	,000	-3,886	16,568	,001	-1:32:08
3	not assumed	25,218	,000	-1,553	3,205	,212	-1:17:33
4	assumed	1,603	,232	-1,955	11,000	,076	-0:25:16
5	assumed	,945	,360	-1,732	8,000	,122	-0:11:30
6	not assumed	7,159	,032	-1,895	2,314	,181	-0:21:49
7	not assumed	,231	,656	-1,133	1,619	,397	-0:28:22

Table 6.30: The t-test results of the comparison between intermediate-experienced individuals and pairs regarding effort. See table 6.29 for the descriptive data. Level one, two and four show a significant difference between individual intermediates and pairing intermediates, favoring the individuals.

have significant mean value differences.

Table 6.31 shows the results of professional individuals and pairs regarding the effort the groups needed to complete tasks (levels). The mean effort needed by professional pairs is higher across all six levels. Level seven has not been completed by a professional team. The number of teams dropped that completed the later levels: e.g. only four teams completed level four (three individuals, one pair). Thus, for the t-test only level one to three will be representative.

Table 6.32 shows the result of the t-tests for table 6.31. Only in level six, the variances are assumed to be not equal according to the Levene tests. Level one and two have a significance value lower than 0.05, which means they are significant: The null hypothesis 3.2.0 can be rejected for these levels. However, the alternative hypothesis cannot be accepted because pairs needed more effort than individuals. The other levels showed no significant differences in the mean values.

Filtered by number of achieved levels The participants can also be grouped by the number of levels the completed in total. Table 6.33 shows the results of participants that solved one, two and three levels. Figure 6.16 shows a graphical representation of this table.
Teams that completed more levels are not shown because they were too few in numbers to be expressive. The upper part of the table represents the participants that completed the first level only. Like in the other effort-related part of this chapter (see the previous pages) the effort needed by pairs was about twice as high as by individuals. This differences proved to be significant, as seen in table 6.34.
Because of removed outliers, the number (N) of data points can vary.

Effort: Individuals vs. Pairs (Professionals)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Effort	1	22	00:56:37	00:29:36	00:06:19
	2	13	01:31:04	00:53:42	00:14:54
Level 2 Effort	1	19	01:07:38	00:39:25	00:09:02
	2	11	02:14:11	01:02:37	00:18:53
Level 3 Effort	1	9	00:39:29	00:27:57	00:09:19
	2	3	01:11:45	00:38:14	00:22:05
Level 4 Effort	1	3	00:52:26	00:21:53	00:12:38
	2	1	03:01:06		
Level 5 Effort	1	2	00:15:39	00:10:36	00:07:30
	2	1	00:51:38		
Level 6 Effort	1	2	00:41:27	00:19:57	00:14:06
	2	1	00:51:38		
Level 7 Effort	1	0			
	2	0			

Table 6.31: Shows the level statistic for professional pairs and individuals regarding effort. Each line shows the results for the particular group in a level: the size of the group, the mean effort, the standard deviation and the standard error mean.

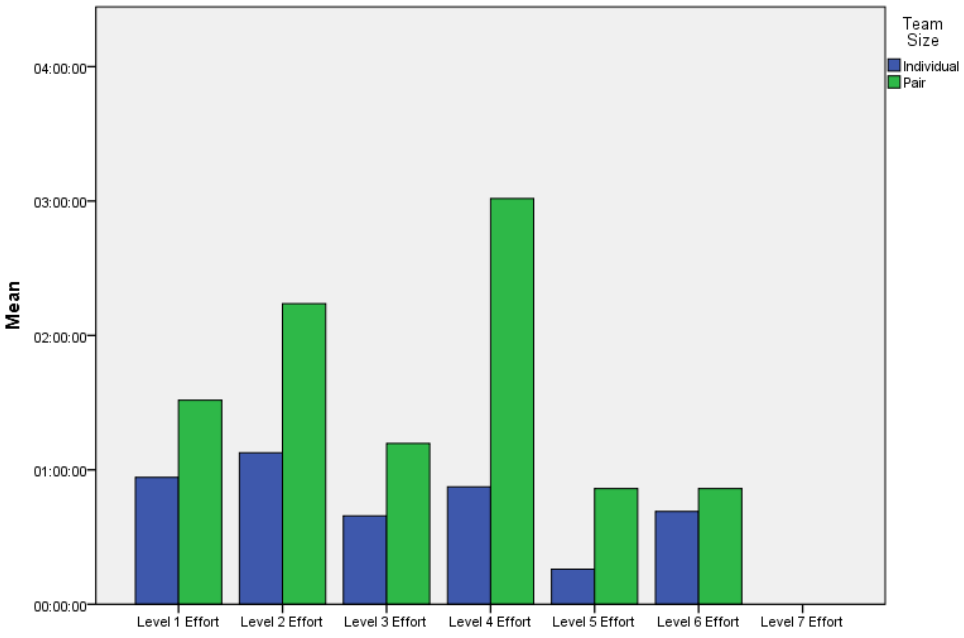


Figure 6.15: Graphical representation of Table 6.31.

T-Test - Effort: Individuals vs. Pairs (Professionals)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	,451	,506	-2,456	33,000	,019	-0:34:26
2	not assumed	6,409	,017	-3,180	14,684	,006	-1:06:33
3	assumed	,377	,553	-1,598	10,000	,141	-0:32:16
4	assumed	.	.	-5,090	2,000	,036	-2:08:39
5	assumed	.	.	-2,772	1,000	,220	-0:35:58
6	assumed	.	.	-,416	1,000	,749	-0:10:10

Table 6.32: The t-test results of the comparison between professional individuals and pairs regarding effort. See table 6.31 for the descriptive data. Level one, two and four show a significant difference between individual professionals and pairing professionals, favoring the individuals.

Effort: Individuals vs. Pairs (Completed Levels)

Number completed levels		Team Size	N	Mean	Std. Deviation	Std. Error Mean
1 Level	Level 1 Effort	1	17	1:24:35	0:25:03	0:06:05
		2	13	2:48:39	1:00:49	0:16:52
2 Levels	Level 1 Effort	1	30	0:51:57	0:22:30	0:04:06
		2	24	1:23:23	0:23:47	0:04:51
	Level 2 Effort	1	31	1:22:55	0:37:45	0:06:47
		2	25	2:52:52	1:22:53	0:16:35
3 Levels	Level 1 Effort	1	16	0:39:17	0:15:46	0:03:57
		2	4	1:17:19	0:12:43	0:06:21
	Level 2 Effort	1	16	0:42:55	0:17:01	0:04:15
		2	4	1:51:18	1:12:04	0:36:02
	Level 3 Effort	1	16	1:00:45	0:37:11	0:09:18
		2	4	2:02:59	0:42:38	0:21:19

Table 6.33: Shows the level statistic for pairs and individuals regarding effort, grouped by the number of levels the completed in total (one to three). Each line shows the results for the particular group in a level: the size of the group, the mean effort, the standard deviation and the standard error mean.

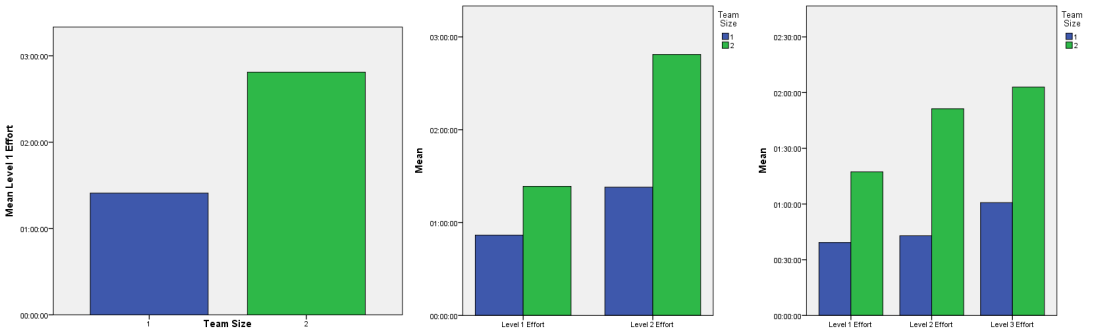


Figure 6.16: Graphical representation of Table 6.33. On the left side, the participants that completed exactly one level, in the middle the teams with two completed levels and on the right side three levels.

T-Test - Effort: Individuals vs. Pairs (All experiences)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	not assumed	10,262	,003	-4,689	15	,000	-1:24:04
1	assumed	,003	,960	-4,974	52	,000	-0:31:26
2	not assumed	15,548	,000	-5,023	32	,000	-1:29:58
1	assumed	,194	,665	-4,446	18	,000	-0:38:02
2	not assumed	12,820	,002	-1,885	3	,153	-1:08:23
3	assumed	,001	,970	-2,918	18	,009	-1:02:14

Table 6.34: The t-test results of the comparison between individuals and pairs that completed one, two or three levels, regarding duration. See table 6.33 for the descriptive data. The t-test shows, that in five of six groups, the difference between pairs and individuals is strongly significant (< 1% error probability).

6.3.3 RI3: Defects

In this subsection, the data for hypothesis 3.3 will be summarized and tested. The hypothesis said, that pairs produce fewer defects than individuals.

Table 6.35 shows the overall results of the experiment regarding defects made of the teams, grouped by individuals and pairs. Teams of three and teams with unknown team sizes are not presented.

The overall results show, that pairs made fewer defects than individuals in five of seven levels. In level six, pairs produced more defects and in level two the numbers were nearly even (slight advantage for individuals).

Table 6.36 shows the summary of the t-tests for the values presented in table 6.35. No significant differences had been found between the mean values of both groups (individuals and pairs). Although, level five shows a 'slight significance' with a value of 0.095 which favours

Defects: Individuals vs. Pairs (All experiences)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1	1	76	1,22	2,195	,252
	2	48	,75	1,263	,182
Level 2	1	60	1,42	2,234	,288
	2	36	1,44	2,049	,341
Level 3	1	29	2,00	2,686	,499
	2	11	1,73	3,003	,905
Level 4	1	13	1,46	2,145	,595
	2	7	1,00	2,236	,845
Level 5	1	9	2,33	3,354	1,118
	2	5	,20	,447	,200
Level 6	1	8	,13	,354	,125
	2	5	,60	,894	,400
Level 7	1	4	1,75	2,062	1,031
	2	2	,50	,707	,500

Table 6.35: Shows the overall (all experience groups together) level statistic for two groups of interest (pairs and individuals) regarding defects. Each line shows the results for the particular group in a level: the size of the group, the mean number of defects made, the standard deviation and the standard error mean.

Pair-Programming. Level one also reached a low (but too high) significance with a value of 0.13.

In total, across all levels there are no significant differences which means that the null hypothesis 3.3.0 cannot be rejected. Ergo, pairs and individuals produce the same amount of defects.

Table 6.37 shows the results of defects made by junior-individuals and junior-pairs. In level one and two, the individuals made fewer mistakes than pairs. In level three and four the pairs were better. Although, the differences are not high and smaller than the standard error mean. The levels four to six were only completed by a few juniors and level seven not at all.

Table 6.38 shows the results of the t-tests made for table 6.37. There are no significant differences between the mean values and their deviations between juniors-individuals and junior-pairs. This means that the null hypothesis 3.3.0 cannot be rejected for juniors.

Table 6.39 shows the results of defects made by intermediate-individuals and intermediate-pairs. The results are mixed since four of seven level favor pairs while the other three favor individuals. The standard error means are higher than the mean differences, except for level five.

Table 6.40 shows the t-tests for the value in table 6.39. There are no significant differences in the mean values and variances between intermediate-individuals and intermediate-pairs. The null hypothesis 3.3.0 cannot be rejected for intermediates.

Table 6.41 shows the defects made by professional-individuals and professional-pairs. The

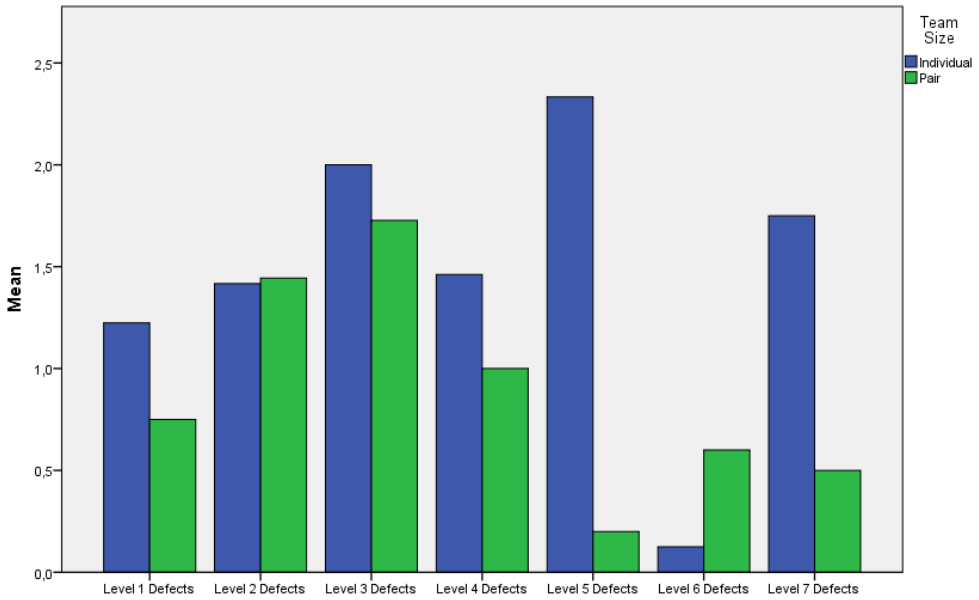


Figure 6.17: Graphical representation of Table 6.35.

T-Test - Defects: Individuals vs. Pairs (All experiences)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	not assumed	8,415	,004	1,524	121,134	,130	,474
2	assumed	,001	,982	-,061	94,000	,952	-,028
3	assumed	,000	,986	,278	38,000	,783	,273
4	assumed	,207	,654	,452	18,000	,656	,462
5	not assumed	4,768	,050	1,878	8,503	,095	2,133
6	not assumed	7,729	,018	-1,133	4,793	,311	-,475
7	not assumed	16,667	,015	1,091	3,926	,338	1,250

Table 6.36: The t-test results of the comparison between individuals and pairs (of all experience levels) regarding defects produced. See table 6.35 for the descriptive data. In level five a slight significant difference was observed, favoring the pairs.

Defects: Individuals vs. Pairs (Juniors)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Defects	1	10	1,10	1,969	0,623
	2	15	1,07	1,438	0,371
Level 2 Defects	1	6	1,17	1,472	0,601
	2	9	1,56	2,351	0,784
Level 3 Defects	1	2	2,00	1,414	1,000
	2	4	0,75	0,957	0,479
Level 4 Defects	1	1	2,00		
	2	2	0,00	0,000	0,000
Level 5 Defects	1	0			
	2	1	0,00		
Level 6 Defects	1	0			
	2	1	0,00		
Level 7 Defects	1	0			
	2	0			

Table 6.37: Shows the level statistic for junior-level pairs and individuals regarding defects made. Each line shows the results for the particular group in a level: the size of the group, the mean number of defects made, the standard deviation and the standard error mean.

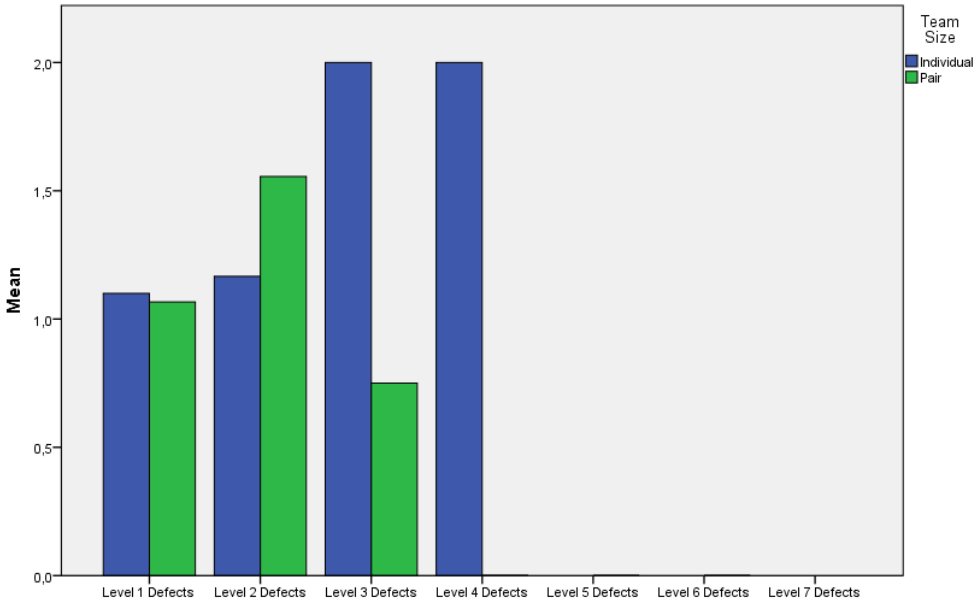


Figure 6.18: Graphical representation of Table 6.37.

T-Test - Defects: Individuals vs. Pairs (Juniors)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	,211	,650	,049	23,000	,961	,033
2	assumed	,624	,444	-,359	13,000	,726	-,389
3	assumed	,667	,460	1,325	4,000	,256	1,250
4	assumed	.	.	.	1,000	.	2,000

Table 6.38: The t-test results of the comparison between junior-level individuals and pairs regarding defects produced. See table 6.37 for the descriptive data. No significant differences were observed.

Defects: Individuals vs. Pairs (Intermediates)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Defects	1	44	1,32	2,280	0,344
	2	20	0,7	1,174	0,263
Level 2 Defects	1	35	1,83	2,706	0,457
	2	16	1,13	1,928	0,482
Level 3 Defects	1	18	2,33	2,521	0,594
	2	4	3,75	4,500	2,250
Level 4 Defects	1	9	1,22	1,922	0,641
	2	4	1,50	3,000	1,500
Level 5 Defects	1	7	3,00	3,559	1,345
	2	3	0,00	0,000	0,000
Level 6 Defects	1	6	0,00	0,000	0,000
	2	3	0,67	1,155	0,667
Level 7 Defects	1	4	1,75	2,062	1,031
	2	2	0,50	0,707	0,500

Table 6.39: Shows the level statistic for intermediate-experienced pairs and individuals regarding defects made. Each line shows the results for the particular group in a level: the size of the group, the mean number of defects made, the standard deviation and the standard error mean.

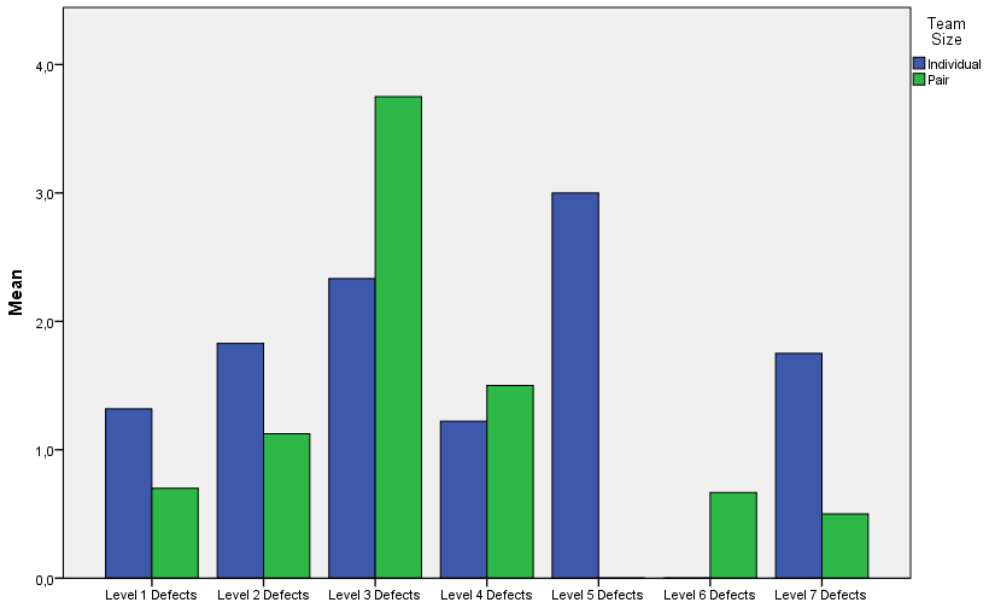


Figure 6.19: Graphical representation of Table 6.39.

T-Test - Defects: Individuals vs. Pairs (Intermediates)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	not assumed	6,796	,011	1,429	60,897	,158	,618
2	assumed	1,696	,199	,935	49,000	,354	,704
3	not assumed	6,037	,023	-,609	3,430	,581	-1,417
4	assumed	1,114	,314	-,204	11,000	,842	-,278
5	assumed	3,738	,089	1,410	8,000	,196	3,000
6	not assumed	37,333	,000	-1,000	2,000	,423	-,667
7	not assumed	16,667	,015	1,091	3,926	,338	1,250

Table 6.40: The t-test results of the comparison between intermediate-experienced individuals and pairs regarding defects produced. See table 6.39 for the descriptive data. No significant differences were observed.

Defects: Individuals vs. Pairs (Professionals)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Defects	1	22	1,09	2,202	0,469
	2	13	0,46	1,198	0,332
Level 2 Defects	1	19	0,74	1,046	0,240
	2	11	1,82	2,089	0,630
Level 3 Defects	1	9	1,33	3,279	1,093
	2	3	0,33	0,577	0,333
Level 4 Defects	1	3	2,00	3,464	2,000
	2	1	1,00		
Level 5 Defects	1	2	0,00	0,000	0,000
	2	1	1,00		
Level 6 Defects	1	2	0,50	0,707	0,500
	2	1	1,00		
Level 7 Defects	1	0			
	2	0			

Table 6.41: Shows the level statistic for professional pairs and individuals regarding effort. Each line shows the results for the particular group in a level: the size of the group, the mean number of defects made, the standard deviation and the standard error mean.

number of professionals is decreasing drastically over the levels to numbers where the results are not representative. E.g. Level four has only four teams that can be compared, level six had only three and no professional participant completed level seven at all. Two of the first three levels favor pairs, while level two favours individuals regarding defects made.

Table 6.42 shows the t-tests for the values of table 6.41. There were no significant differences in the mean values and deviations of defects produced between professional-individuals and pairs. There is no t-value and significance for level five because the mean, standard deviation and standard error mean is zero for individuals and there was only one data row for pairs.

Filtered by number of achieved levels The participants can also be grouped by the number of levels the completed in total. Table 6.43 shows the results of participants that solved one, two and three levels. Figure 6.21 shows a graphical representation of this table.
Teams that completed more levels are not shown because they were too few in numbers to be expressive. The first two groups (1 and 2 completed levels) show, that pairs made less defects than individuals. In one case (2 completed levels, first level) these changes are significant as seen in table 6.44.
The findings for teams that completed three levels are mixed: In the first two levels the individuals produces fewer defects, in level three the pairs.
Because of removed outliers, the number (N) of data points can vary.

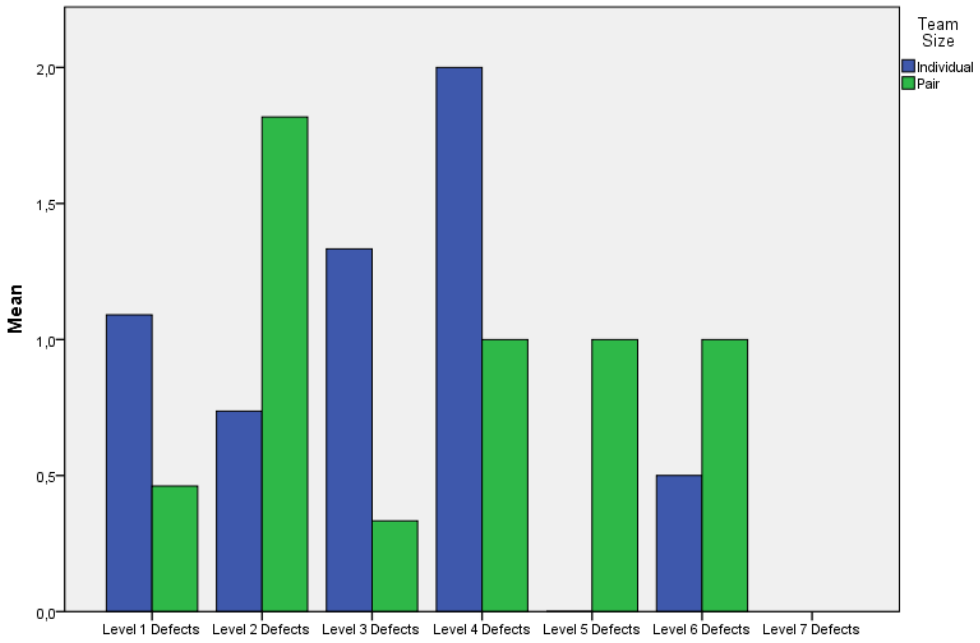


Figure 6.20: Graphical representation of Table 6.41.

T-Test - Defects: Individuals vs. Pairs (Professionals)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	2,372	,133	,947	33,000	,350	0,629
2	not assumed	5,558	,026	-1,604	12,961	,133	-1,081
3	assumed	,937	,356	,510	10,000	,621	1,000
4	assumed	.	.	,250	2,000	,826	1,000
5	assumed	.	.	.	1,000	.	-1,000
6	assumed	.	.	-,577	1,000	,667	-0,500

Table 6.42: The t-test results of the comparison between professional individuals and pairs regarding defects produced. See table 6.41 for the descriptive data. No significant differences were observed.

Defects: Individuals vs. Pairs (Completed Levels)

Number completed levels		Team Size	N	Mean	Std. Deviation	Std. Error Mean
1 Level	Level 1 Defects	1	17	1,9	2,6	,6
		2	13	1,4	1,4	,4
2 Levels	Level 1 Defects	1	30	1,4	2,3	,4
		2	24	,5	1	,2
	Level 2 Defects	1	31	1,9	2,5	,5
		2	25	1,4	2,1	,4
3 Levels	Level 1 Defects	1	16	,6	1,5	,4
		2	4	,8	1,5	,8
	Level 2 Defects	1	16	,6	1,3	,3
		2	4	2,3	2,9	1,4
	Level 3 Defects	1	16	1,3	2,1	,5
		2	4	,5	,6	,3

Table 6.43: Shows the level statistic for pairs and individuals regarding defects, grouped by the number of levels the completed in total (one to three). Each line shows the results for the particular group in a level: the size of the group, the mean number of defects, the standard deviation and the standard error mean.

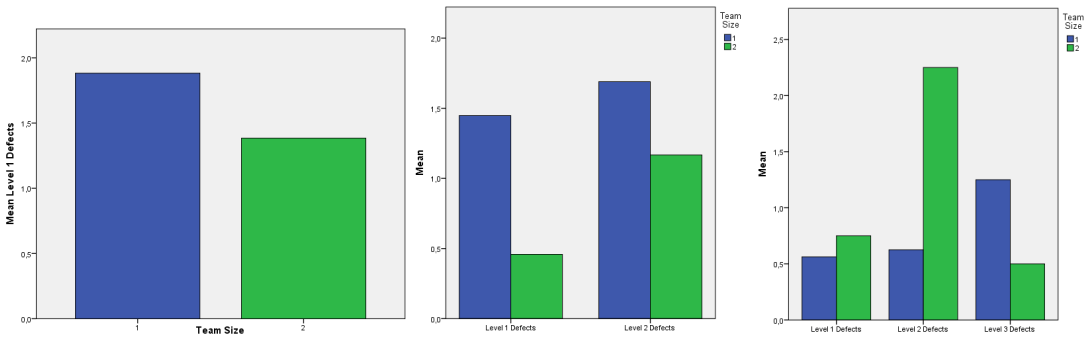


Figure 6.21: Graphical representation of Table 6.43. On the left side, the participants that completed exactly one level, in the middle the teams with two completed levels and on the right side three levels.

T-Test - Defects: Individuals vs. Pairs (All experiences)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	not assumed	5,495	,026	,663	26	,513	,4977
1	not assumed	11,429	,001	2,076	41	,044	,9750
2	assumed	,193	,662	,854	54	,397	,5432
1	assumed	,051	,823	-,218	18	,830	-,1875
2	not assumed	6,043	,024	-1,103	3	,343	-1,6250
3	assumed	1,266	,275	1,232	18	,234	,7500

Table 6.44: The t-test results of the comparison between individuals and pairs that completed one, two or three levels, regarding duration. See table 6.44 for the descriptive data. The second row shows a significant difference, that pairs produced fewer defects than individuals.

6.3.4 RI4: Duration

In this subsection, the data for hypothesis 4.1 will be summarized and tested. The hypothesis said, that highly experienced participants (professionals) complete tasks faster than less experienced participants (juniors).

Participants with intermediate experience have not been evaluated.

Table 6.45 shows the duration results of individuals and pairs grouped by juniors and professionals. Teams of three, teams with unknown sizes are excluded.

In the results, the first levels (with a higher number of data points) show that professionals, on average, solved tasks faster than juniors. In the later levels (four, five and six) juniors were faster than professionals, but this probably resulted from an excellent junior participant.

Table 6.46 shows the t-tests of the values presented in table 6.45. For all levels, the Levene tests suggested that the variances of the groups (juniors and professionals) are equal. Level three shows a significant difference in the duration needed to complete a level with a value of 0.03. On average, professionals completed this level 32 minutes and 26 seconds faster than juniors. This means that for this level the null hypothesis 4.1.0 can be rejected and the alternative 4.1.1 accepted.

The other levels showed no significant differences.

Table 6.47 shows the results like in table 6.45 but only took individual participants into account. In two levels, the results were approximately even (level one and three), in two levels the professionals were faster than juniors (level two and four). Level five and six didn't allow a comparison because no junior completed these levels. Level seven wasn't completed by any of them.

Table 6.48 shows the results of the t-tests made for the values in table 6.47. According to the Levene tests, the variances of both groups are equal in all four levels. There have been no significant differences between the mean values for junior-individuals and professional-individuals. Hypothesis 4.3.0 cannot be rejected in this group.

Duration: Juniors vs. Professionals (Individuals and Pairs)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Duration	Junior	25	00:56:14	00:26:01	00:05:12
	Professional	35	00:52:30	00:28:43	00:04:51
Level 2 Duration	Junior	15	01:22:26	00:49:35	00:12:48
	Professional	30	01:07:26	00:36:04	00:06:35
Level 3 Duration	Junior	6	01:11:01	00:30:56	00:12:37
	Professional	12	00:38:35	00:25:14	00:07:17
Level 4 Duration	Junior	3	00:51:39	00:30:17	00:17:29
	Professional	4	01:01:58	00:26:07	00:13:03
Level 5 Duration	Junior	1	00:08:19	.	.
	Professional	3	00:19:02	00:09:30	00:05:29
Level 6 Duration	Junior	1	00:10:48	.	.
	Professional	3	00:36:14	00:16:45	00:09:40
Level 7 Duration	Junior	0	.	.	.
	Professional	0	.	.	.

Table 6.45: Shows the overall (pairs and individuals) level statistic for two groups of interest (juniors and professionals) regarding duration. Each line shows the results for the particular group in a level: the size of the group, the mean duration, the standard deviation and the standard error mean.

T-Test - Duration: Juniors vs. Professionals (Individuals and Pairs)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	,245	,622	,516	58,000	,608	00:03:44
2	assumed	3,872	,056	1,158	43,000	,253	00:14:59
3	assumed	,413	,530	2,389	16,000	,030	00:32:26
4	assumed	,102	,762	-,485	5,000	,648	-00:10:19
5	assumed	.	.	-,976	2,000	,432	-00:10:43
6	assumed	.	.	-1,315	2,000	,319	-00:25:26

Table 6.46: The t-test results of the comparison between junior-experienced and professionals (both team sizes together) regarding duration. See table 6.45 for the descriptive data. In level three a significant difference was observed, favoring the professionals.

Table 6.49 shows the duration results for junior-pairs and professional-pairs. In the first three levels, professionals were faster than juniors. In level three, professional were faster by 140%.

In level four to six the juniors were faster but the number of data points was very low (lower than four) in this levels.

Table 6.50 shows the results of the t-tests made for the values in table 6.49. The Levene tests

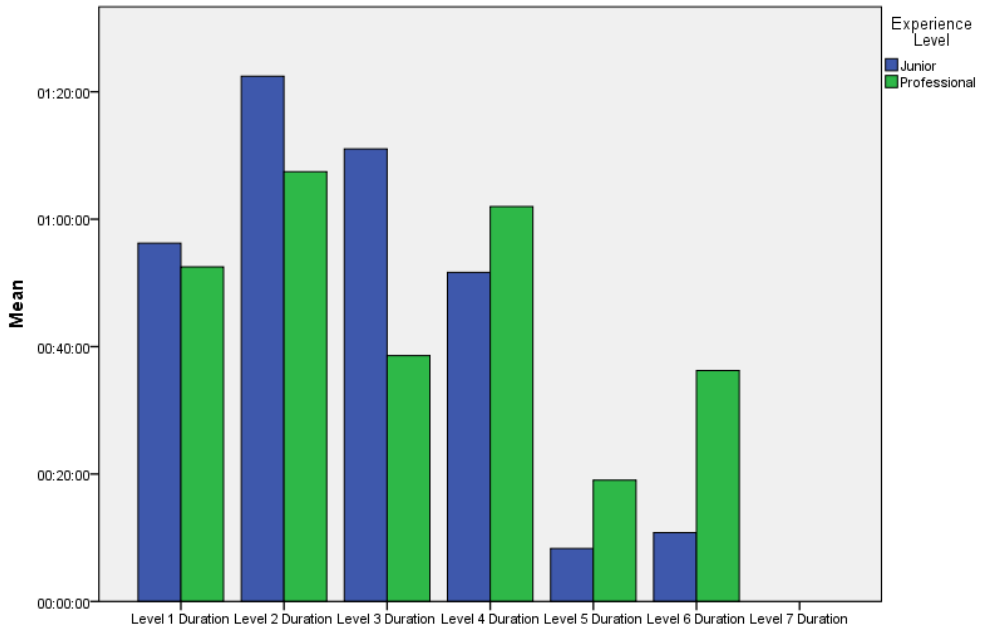


Figure 6.22: Graphical representation of Table 6.45.

suggested that the variances were not equal in level two and equal in the other levels. Level three shows a significant difference of the mean, deviation and standard error between both groups by a value of 0.023. For this level, the null hypothesis 4.1.0 can be rejected and the alternative hypothesis 4.1.1 accepted. Professional-pairs are faster than junior-pairs.

The other levels showed no significant difference. Level five and six showed no values at all because only one pair of each group completed these levels.

6.3.5 RI4: Defects

In this subsection, the data for hypothesis 4.2 will be summarized and tested. The hypothesis said, that highly experienced participants (professionals) produce fewer defects than less experienced participants (juniors).

Participants with intermediate experience have not been evaluated.

Table 6.51 shows the defects results of individuals and pairs grouped by juniors and professionals. Teams of three, teams with unknown sizes are excluded.

In the results, professionals made fewer defects than juniors in the first three levels but the deviation and standard error of the mean show, that these differences are small. In the other levels (four to six) the juniors made fewer defects but the number of participants that completed these levels is low. Level seven was not completed by any of this group of subjects.

Table 6.52 shows the results of the t-tests made for the values in table 6.51. The Levene tests suggest that the variances of both groups are equal for all levels. The analysis of the mean values showed no significant differences across all levels.

Duration: Juniors vs. Professionals (Individuals)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Duration	Junior	10	00:54:58	00:20:22	00:06:26
	Professional	22	00:56:37	00:29:36	00:06:18
Level 2 Duration	Junior	6	01:28:09	00:50:38	00:20:40
	Professional	19	01:07:38	00:39:24	00:09:02
Level 3 Duration	Junior	2	00:39:08	00:18:11	00:12:51
	Professional	9	00:39:29	00:27:56	00:09:18
Level 4 Duration	Junior	1	01:11:26	.	.
	Professional	3	00:52:26	00:21:53	00:12:38
Level 5 Duration	Junior	0	.	.	.
	Professional	2	00:15:39	00:10:35	00:07:29
Level 6 Duration	Junior	0	.	.	.
	Professional	2	00:41:27	00:19:57	00:14:06
Level 7 Duration	Junior	0	.	.	.
	Professional	0	.	.	.

Table 6.47: Shows the level statistic for junior and professional individuals regarding duration. Each line shows the results for the particular group in a level: the size of the group, the mean duration, the standard deviation and the standard error mean.

T-Test - Duration: Juniors vs. Professionals (Individuals)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	3,171	,085	-,161	30,000	,874	-00:01:39
2	assumed	,603	,445	1,040	23,000	,309	00:20:30
3	assumed	,421	,533	-,016	9,000	,987	-00:00:20
4	assumed	.	.	,751	2,000	,531	00:18:59

Table 6.48: The t-test results of the comparison between junior-experienced and professional individuals regarding duration. See table 6.47 for the descriptive data. No significant differences observed.

Table 6.53 shows the defects made by junior-individuals and professional-individuals. In two levels (two and three) professionals made fewer defects than juniors, although the standard error of the mean was high so that these results are not representative (as shown later in table 6.54). In level one and four the average number of defects was approximately equal. No junior completed level five or six so that no comparison can be made.

Table 6.54 shows the results of the t-tests made for the values in table 6.53. The Levene tests suggest that the variances of both groups are equal for all levels. No significant differences have been found between junior-individuals and professional-individuals.

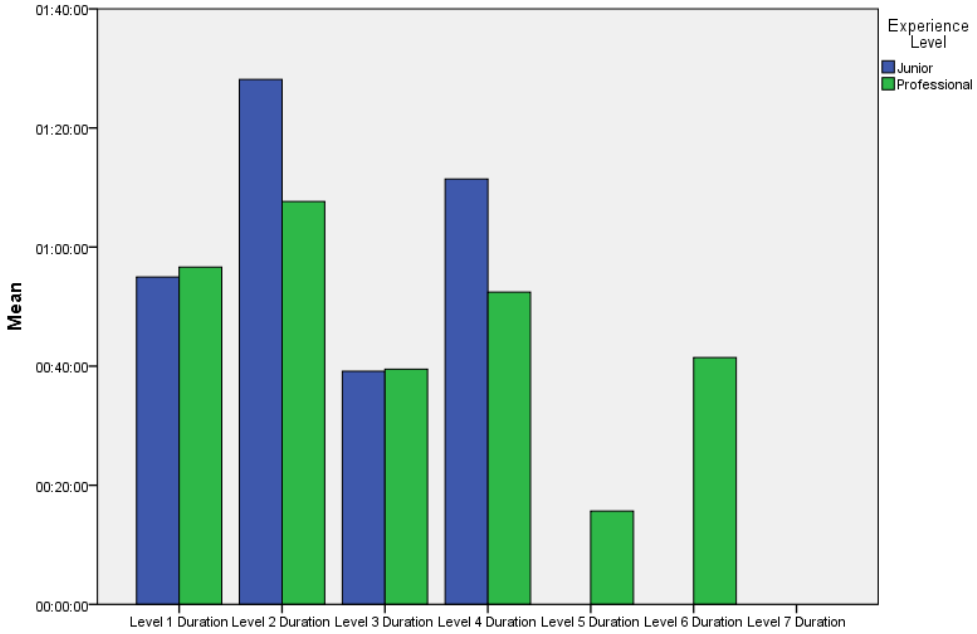


Figure 6.23: Graphical representation of Table 6.47.

Table 6.55 shows the results of defects made by junior-pairs and professional-pairs. In level one and three, the professionals made fewer defects than juniors. In level two the juniors produced fewer defects on average. The standard errors of the means show, that the mean values can still vary more than the differences of the means. Level four to six are not representative because the number of subjects in these levels is too low.

Table 6.56 shows the results of the t-tests made for the values in table 6.55. The Levene tests suggest that the variances of both groups are equal for all levels. No significant differences have been found across all levels between juniors and professionals.

Duration: Juniors vs. Professionals (Pairs)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Duration	Junior	15	00:57:05	00:29:52	00:07:42
	Professional	13	00:45:32	00:26:50	00:07:26
Level 2 Duration	Junior	9	01:18:37	00:51:34	00:17:11
	Professional	11	01:07:05	00:31:18	00:09:26
Level 3 Duration	Junior	4	01:26:57	00:21:39	00:10:49
	Professional	3	00:35:52	00:19:07	00:11:02
Level 4 Duration	Junior	2	00:41:45	00:35:19	00:24:58
	Professional	1	01:30:33	.	.
Level 5 Duration	Junior	1	00:08:19	.	.
	Professional	1	00:25:49	.	.
Level 6 Duration	Junior	1	00:10:48	.	.
	Professional	1	00:25:49	.	.
Level 7 Duration	Junior	0	.	.	.
	Professional	0	.	.	.

Table 6.49: Shows the level statistic for junior and professional pairs regarding duration. Each line shows the results for the particular group in a level: the size of the group, the mean duration, the standard deviation and the standard error mean.

T-Test - Duration: Juniors vs. Professionals (Pairs)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	1,331	,259	1,070	26,000	,295	00:11:33
2	not assumed	4,596	,046	,588	12,633	,567	00:11:31
3	assumed	,037	,855	3,234	5,000	,023	00:51:05
4	assumed	.	.	-1,128	1,000	,462	-00:48:47
5	assumed	.	.	.	,000	.	-00:17:30
6	assumed	.	.	.	,000	.	-00:15:01

Table 6.50: The t-test results of the comparison between junior-experienced and professional pairs regarding duration. See table 6.49 for the descriptive data. In level three a significant (0.023) difference was observed, favoring the professionals.

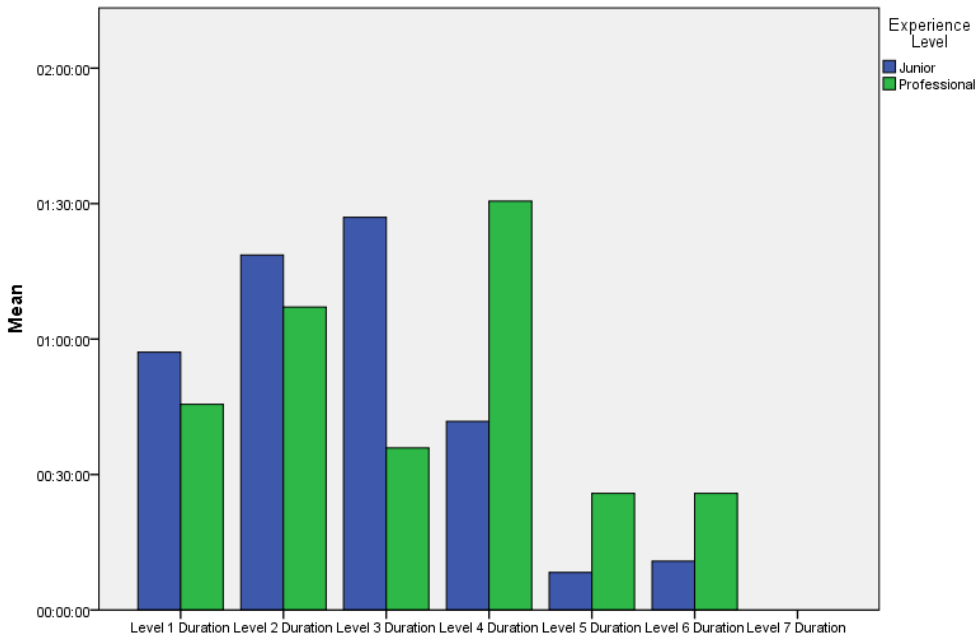


Figure 6.24: Graphical representation of Table 6.49.

Defects: Juniors vs. Professionals (Individuals and Pairs)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Defects	Junior	25	1,08	1,631	,326
	Professional	35	,86	1,896	,321
Level 2 Defects	Junior	15	1,40	1,993	,515
	Professional	30	1,13	1,570	,287
Level 3 Defects	Junior	6	1,17	1,169	,477
	Professional	12	1,08	2,843	,821
Level 4 Defects	Junior	3	,67	1,155	,667
	Professional	4	1,75	2,872	1,436
Level 5 Defects	Junior	1	,00	.	.
	Professional	3	,33	,577	,333
Level 6 Defects	Junior	1	,00	.	.
	Professional	3	,67	,577	,333
Level 7 Defects	Junior	0	.	.	.
	Professional	0	.	.	.

Table 6.51: Shows the overall (pairs and individuals) level statistic for two groups of interest (juniors and professionals) regarding defects made. Each line shows the results for the particular group in a level: the size of the group, the mean number of defects made, the standard deviation and the standard error mean.

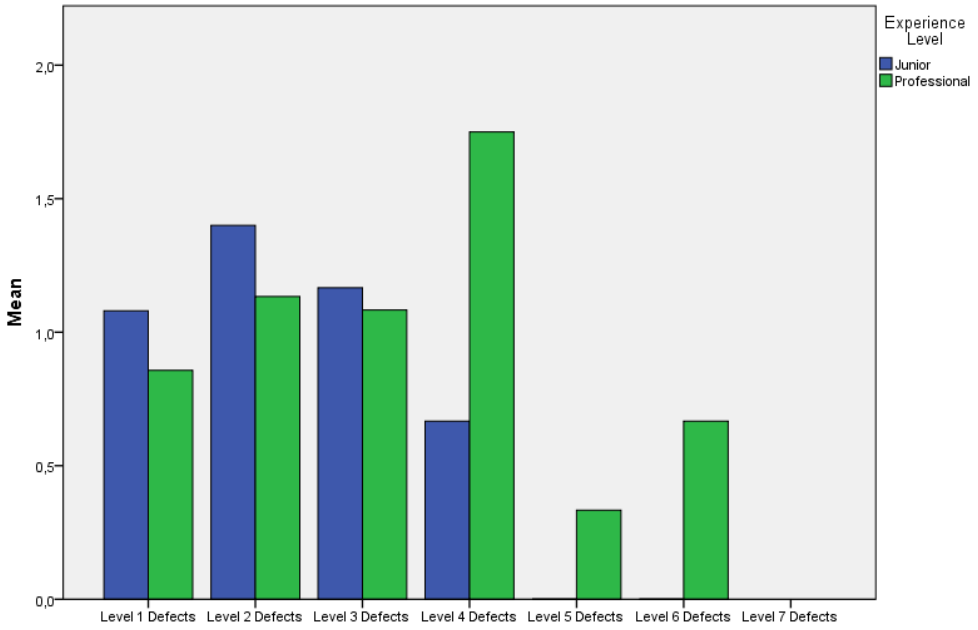


Figure 6.25: Graphical representation of Table 6.51.

T-Test - Defects: Juniors vs. Professionals (Individuals and Pairs)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	,009	,923	,475	58,000	,637	,223
2	assumed	1,139	,292	,491	43,000	,626	,267
3	assumed	,354	,560	,068	16,000	,947	,083
4	assumed	1,875	,229	-,606	5,000	,571	-1,083
5	assumed	.	.	-,500	2,000	,667	-,333
6	assumed	.	.	-1,000	2,000	,423	-,667

Table 6.52: The t-test results of the comparison between junior-experienced and professionals (both team sizes together) regarding defects produced. See table 6.51 for the descriptive data. No significant differences were observed.

Defects: Juniors vs. Professionals (Individuals)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Defects	Junior	10	1,10	1,969	,623
	Professional	22	1,09	2,202	,469
Level 2 Defects	Junior	6	1,17	1,472	,601
	Professional	19	,74	1,046	,240
Level 3 Defects	Junior	2	2,00	1,414	1,000
	Professional	9	1,33	3,279	1,093
Level 4 Defects	Junior	1	2,00	.	.
	Professional	3	2,00	3,464	2,000
Level 5 Defects	Junior	0	.	.	.
	Professional	2	,00	,000	,000
Level 6 Defects	Junior	0	.	.	.
	Professional	2	,50	,707	,500
Level 7 Defects	Junior	0	.	.	.
	Professional	0	.	.	.

Table 6.53: Shows the level statistic for junior and professional individuals regarding defects made. Each line shows the results for the particular group in a level: the size of the group, the mean number of defects made, the standard deviation and the standard error mean.

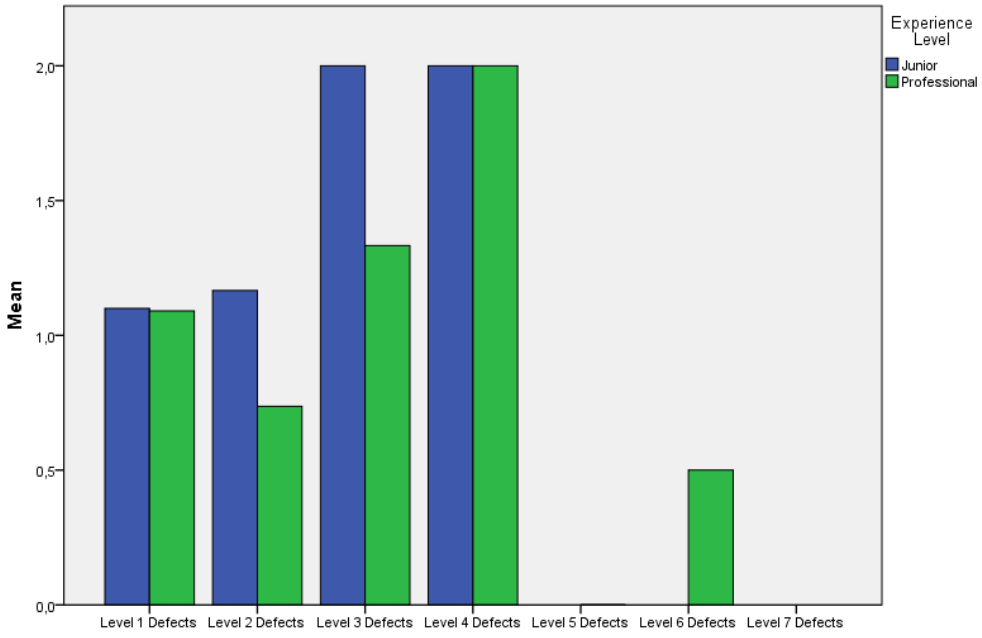


Figure 6.26: Graphical representation of Table 6.53.

T-Test - Defects: Juniors vs. Professionals (Individuals)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	,071	,792	,011	30,000	,991	,009
2	assumed	1,849	,187	,797	23,000	,434	,430
3	assumed	,240	,636	,273	9,000	,791	,667
4	assumed	.	.	,000	2,000	1,000	,000

Table 6.54: The t-test results of the comparison between junior-experienced and professional individuals regarding defects produced. See table 6.53 for the descriptive data. No significant differences were observed.

Defects: Juniors vs. Professionals (Pairs)

	Team Size	N	Mean	Std. Deviation	Std. Error Mean
Level 1 Defects	Junior	15	1,07	1,438	,371
	Professional	13	,46	1,198	,332
Level 2 Defects	Junior	9	1,56	2,351	,784
	Professional	11	1,82	2,089	,630
Level 3 Defects	Junior	4	,75	,957	,479
	Professional	3	,33	,577	,333
Level 4 Defects	Junior	2	,00	,000	,000
	Professional	1	1,00	.	.
Level 5 Defects	Junior	1	,00	.	.
	Professional	1	1,00	.	.
Level 6 Defects	Junior	1	,00	.	.
	Professional	1	1,00	.	.
Level 7 Defects	Junior	0	.	.	.
	Professional	0	.	.	.

Table 6.55: Shows the level statistic for junior and professional pairs regarding defects made. Each line shows the results for the particular group in a level: the size of the group, the mean number of defects made, the standard deviation and the standard error mean.

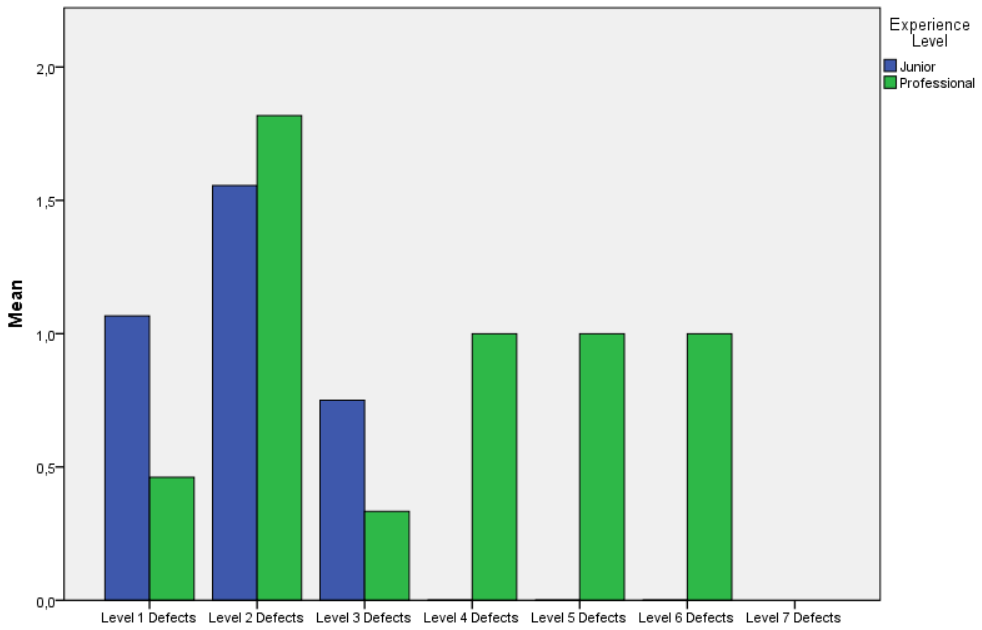


Figure 6.27: Graphical representation of Table 6.55.

T-Test - Defects: Juniors vs. Professionals (Pairs)

		Levene's Test for Equality of Variances		t-test for Equality of Means			
Level	Equal variance	F	Sig.	t	df	Sig.	Mean Diff.
1	assumed	1,487	,234	1,198	26,000	,242	,605
2	assumed	,033	,859	-,264	18,000	,794	-,263
3	assumed	1,394	,291	,660	5,000	,538	,417
4	assumed	.	.	.	1,000	.	-1,000
5	assumed	.	.	.	,000	.	-1,000
6	assumed	.	.	.	,000	.	-1,000

Table 6.56: The t-test results of the comparison between junior-experienced and professional pairs regarding defects produced. See table 6.55 for the descriptive data. No significant differences were observed.

CHAPTER 7

Discussion

In this chapter a brief summary of the topic and the research methodology will be given. Followed by four sections that discuss each research issues.

The aim of this master thesis is to analyze coding contests how they can be used for empirical studies, and secondly, a pilot application in which a coding contest is conducted and its result used to support the study of topics under investigation in the field of computer science.

An experiment was embedded into a coding contest which took place in October 2012. The participants of this coding contest are tested on how pairs competed against individuals and how experience affected their efficiency. The knowledge gained from the experiment shall be used to describe guidelines for future researchers about how to plan and embed their own experiments into coding contests.

Further, it will be discussed how coding-contest-experiments can be used in an empirical ecosystem. If the pilot application proves to be effective, i.e. the results of the experiments are representative and useful, then it will be evaluated how coding contests can be used to support a body of knowledge.

The experiment/pilot application itself consists of five hypotheses which will be tested based on the results using Student's t-test. The focus of the hypotheses lays on pair vs. single programming and effects of experience on coding efficiency. These two fields are currently under research so that the experiment's results can be easily approved.

7.1 Coding contests as vehicle for empirical research (Research Topic 1)

In this section, RI1 and RI2 are discussed. RI1 addresses the practical part while RI2 focuses on the scientific point of view.

7.1.1 How can coding contests be set up to support controlled experiments? (RI1)

In chapter 5 the experiment setup and execution was presented. In this subsection the experiences about the conduction of the embedded experiment are discussed.

Participant information The first thing to mention is the provided information about the participants. Right now the data collected during the registration process and the Catcoder tool is substantial but the validity has to be improved. As described in the results chapter 6.1.1, many data points had to be excluded because the team size was unknown (due to incorrect registration data). Secondly, the experience level (junior, intermediate, professional) was chosen by the participants themselves which means that many of them are overrated or underrated. To compensate these issues, the subjects were also asked, on a free basis, for their years of experience in programming. Comparing these numbers to the rating of the three experience levels showed, that the majority showed reasonable results as shown in figure 7.1. Notice, that this boxplot is based on 142 of 173 data points because 31 teams did not submit their experience. Further, it represents only one team member of pairs and teams of three.

However, there were also exceptions like intermediates with 20 years of programming experience which suggests that future experiments should extend the subject data collection process.

A researcher that uses data from coding contests should also consider that some participants have exceptional skills in relation to their age or years of coding experience. At the CCC experiment there were at least three former, respectively active, IOI end-round contestants. Due to their young age they could be rated as juniors and thereby distort the research data.

Conclusion: All participants should be rated about their experience level by the experiment designer to improve the data consistency. Further, the submitted personal data (including team size) should be controlled again on the experiment day for validity reasons. (Teams can change in the last minutes before the contest)

Simple and complex problems As described in chapter 5 the experiment had a level based design. Level three to seven were more complex while level one and two were basic tasks. In this pilot application 44 teams solved level one only, 81 only the first two and 28 the first three levels in the given time limit (out of 173 teams). Only 6 out of 173 completed all seven levels. The design, that only a small percentage of teams should solve all levels is by intention of the experiment hoster Catalysts GmbH to give every participant a challenge.

However, the small number of teams that solved a complex level limits the benefit for researchers, because the majority of the teams can only be compared based on their work on simple tasks.

Conclusion: The levels should be designed in a way that the majority of the participating teams solve at least one complex and one simple task. A solution could be an alternating design, e.g. simple, complex, simple, complex, etc.

Level based design The level based design was used so that contestants can attend alone without having a disadvantage compared to pairs or teams of three. Because only one task is

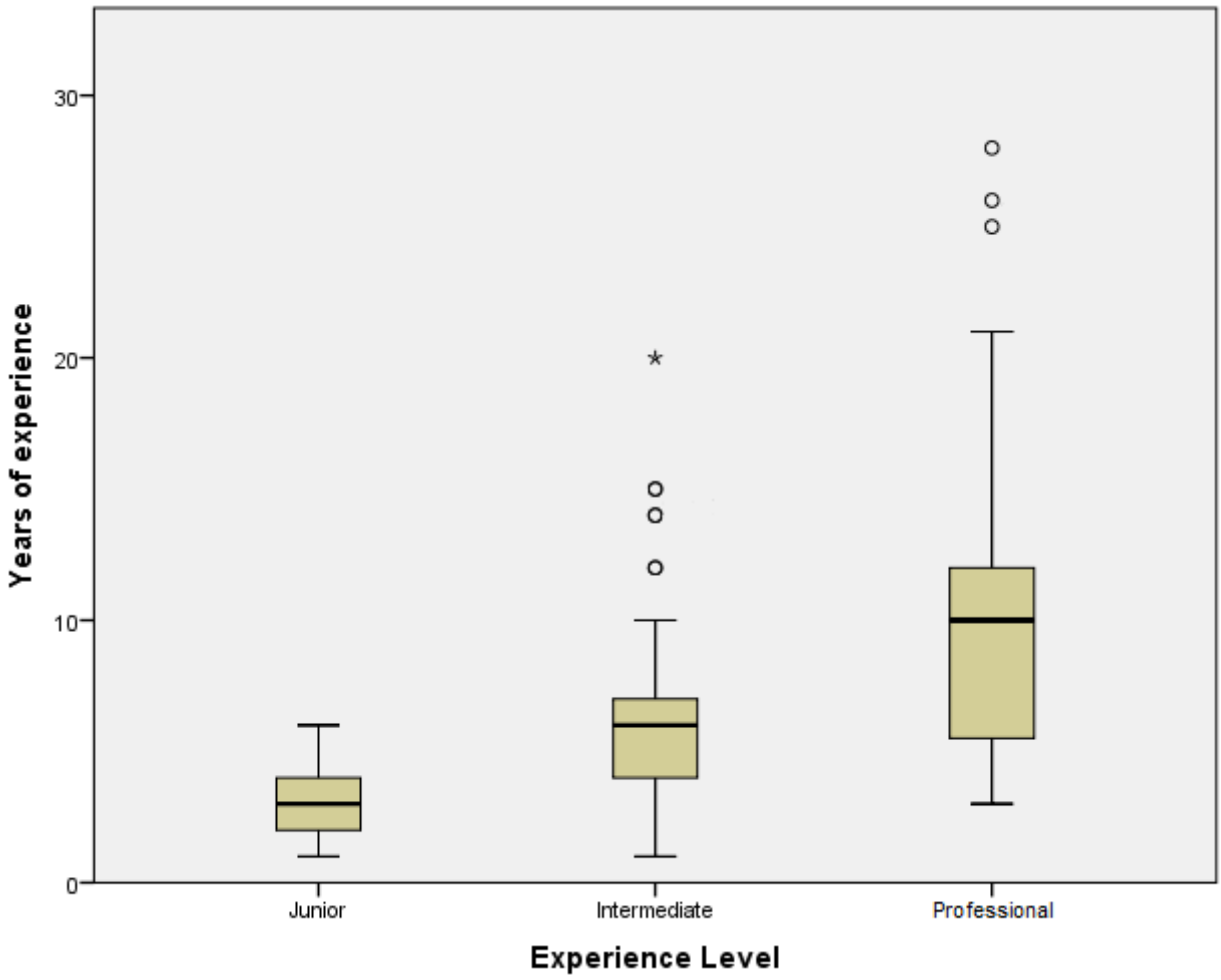


Figure 7.1: Boxplot that shows how the participants rated themselves (junior, intermediate, professional) and how many years of experience they have in programming. Pairs and teams of three are represented by only one data point.

visible at a time, pairs and bigger groups cannot split up to prepare other tasks in parallel to programming the current task (like it is standard at the ICPC and Challenge24).

The results of the experiment showed, that single programmers did not compete weaker than pairs. They even indicate that they competed better than pairs.

Conclusion: If researchers want to compare teams of different sizes against each other, then the CCC would suit to this requirement. Other studies, e.g. Arisholm et al. [3] compared the results of two experiments that were about four years apart. First they had solos as subjects, later pairs. When the experiment is split into parts there is always the chance that the results are not comparable because working environments, programming languages, working procedures, etc. could have evolved strongly enough to cause a significant advantage for the subjects in the chronologically later experiments.

7.1.2 How can the concept of coding contests support research and replication in the field of empirical software engineering? (RI2)

Systematical and Repeatable Since the CCC is currently held thrice per year and will continue to do so for the foreseeable future it can be used to create replications of former experiments without much effort. An experiment conductor can use this to repeat the same experiment multiple times or change a single variable to test its effect. The high repetition number allows it to systematically try out the effects of variable multiple times per year. It saves the researcher a lot of time if one doesn't have to perform and host all the experiments by him/herself.

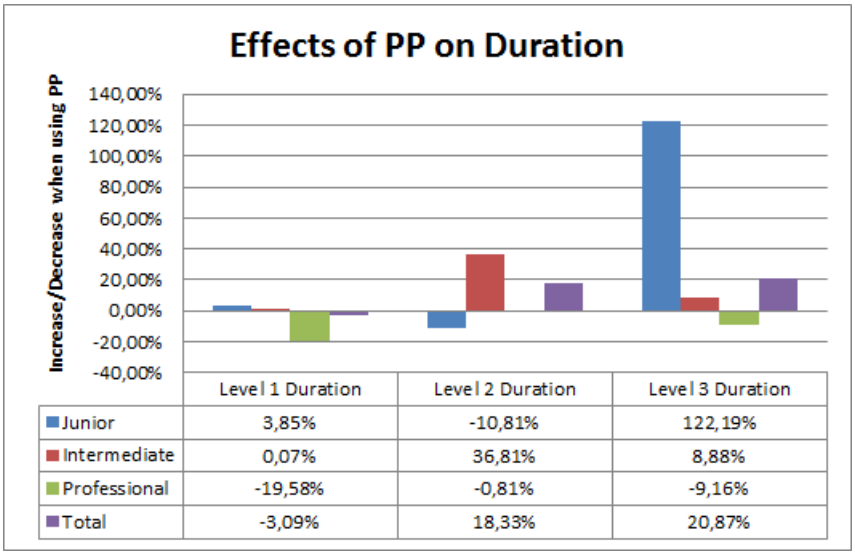
Heterogeneous The wide background distribution of the subjects in the CCC and the large participants numbers allow it to compare the different groups to each other in a single experiment as well as over multiple ones. Due the high number of participants the researcher can also only select the subject groups he is interested in and still have enough data points to work with.

Participant distributions show that over 20 % are juniors and professionals and around 50 % intermediates. Since many experiments in the literature focus on students, the high number of professionals can provide a useful source of information. The CCC can be adopted to host experiments that are only focused on a single participation group.

Common data model Biffl et al. [11] suggested a common data model for experiments so that every researcher can read ones experiment results through a standardized interface. The CCC can fulfill this requirement by applying the interface before publishing their data.

7.2 Empirical evaluation of software-development-techniques effects in context of coding contests (Research Topic 2)

This section discusses RI3, which focuses on the effects of Pair-Programming, and RI, which addresses the effects of experience on coding efficiency.



	Level 1 Duration	Level 2 Duration	Level 3 Duration
Junior	3,85%	-10,81%	122,19%
Intermediate	0,07%	36,81%	8,88%
Professional	-19,58%	-0,81%	-9,16%
Total	-3,09%	18,33%	20,87%

Figure 7.2: Bar chart that shows the percentage difference between pair-programming teams and individual-programmers regarding the time they needed to complete a task. A positive number represents an increase of time needed when using Pair-Programming. Only the first three levels are shown because the later levels are not as representative due less data points.

7.2.1 What are the effects of developing software in teams? (RI3)

Effects of Pair-Programming on programming efficiency regarding duration Figure 7.2 shows the relation of duration-efficiency. The bar chart shows three groups, representing the level one, two and three of the coding contest. The number of participants that solved level four was declining severe so the results of these levels are not as representative as the first levels. Each of the three groups has four bars in different colors, showing the three particular experience groups junior, intermediate and professionals as well as a total overview.

Level one does not show considerable differences between pairs and singles except for professionals, where professional-pairs worked around 20 % faster than professional-individuals.

In level two, the junior pairs were about 11 % faster than individual juniors and intermediate pairs were 37 % slower than their counterpart.

In level three, the time junior pairs needed to complete the task increased by 122 %. Reasons for this increase could be that juniors lack the self confidence to attend alone, so that the higher proportion participate in a team and a few very skilled and confident juniors attend alone. Intermediate pairs needed 9 % more time than individuals, while professionals pairs decreased their duration by 9 %.

Comparison to literature results When comparing the results from the experiment to the summary of the literature research in chapter 2, then there is less accordance between them. Table 2.1 summarizes the literature results and figure 7.2 the CCC experiment. The studies of the literature research suggested a mean duration advantage of pairs of 40 %.

Exerience level	Complexity	CCC Experiment	Arisholm 2007 Expermiment
Overall	All	12%	-8%
	Simple	8%	-20%
	Complex	21%	6%
Juniors	All	38%	5%
	Simple	-3%	4%
	Complex	122%	6%
Intermediates	All	15%	-28%
	Simple	18%	-39%
	Complex	9%	-16%
Professionals	All	-10%	-9%
	Simple	-10%	-23%
	Complex	-9%	8%

Table 7.1: Comparison of this thesis' experiment and the Arisholm 2007 [3] experiments results about the effects of programming in pairs regarding effort. The numbers show the increase and decrease of time team needed when using pair-programming.

In the CCC experiment, pairs had a speed disadvantage of 12 % for completing tasks. This means that pairs were slower than individuals which do not correspond with the literature experiments. Only two other experiments, mentioned in Hannay et al. [29], showed a negative effect on duration for pairs. However, Hannay et al. [29] also state that there seem to be a publication bias because there seem to be too few published studies that show negative results for PP.

Arisholm et al. [3] only showed an overall 8 % advantage for PP on duration which is the closest match from the studies reviewed in the related work chapter 2. Further, they also split the results into juniors, intermediates and professionals as well as complex and simple tasks (see figure in their other study [29]).

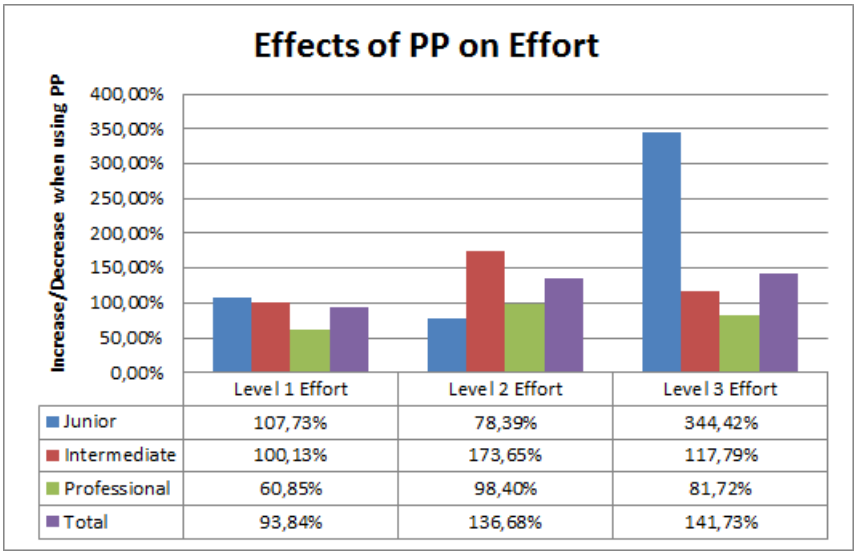
Figure 7.1 compares the results of Arisholm et al. and this master thesis experiment. The results differ in many rows, only the professionals' results seem to be related.

Discussion There have been no signals, that Pair-Programming is faster than individual programming. In fact, in two cases the individuals were slightly-significant faster than pairs: a) Junior-singles vs. junior-pairs with a significance of 0,057 and b) Intermediate-singles vs. intermediate-pairs with a significance of 0.082.

Since there were no significant results when looking at the time needed to complete tasks, the null hypothesis 3.1.0 (No difference between pairs and singles) cannot be rejected. However, there seem to be an advantage towards the individuals.

Effects of Pair-Programming on programming efficiency regarding effort Figure 7.3 shows the effect of PP on effort needed to solve the first three levels, grouped by the experience level of the participants.

Effort is calculated by multiplying duration with team size, thus the numbers represent the man-hours or cost of Pair-Programming and single programming.



	Level 1 Effort	Level 2 Effort	Level 3 Effort
Junior	107,73%	78,39%	344,42%
Intermediate	100,13%	173,65%	117,79%
Professional	60,85%	98,40%	81,72%
Total	93,84%	136,68%	141,73%

Figure 7.3: Bar chart that shows the effort (person hours) needed by individual programmers and pair programmers to complete the first three tasks of the experiment. This chart is similar to figure 7.2 but the duration needed by pairs was doubled in order to obtain the person hours.

The effort is higher when using PP across all experience levels and coding contest levels.

In level one the additional effort was around 100 % for juniors and intermediates and only 60 % for professionals.

In level two the additional effort needed by juniors decreased to 78 %, intermediates increased to 173 % and professionals at 98 %.

In level three the additional effort of juniors jumped up to 344 %, intermediates needed 117 % and professionals 82 %.

Comparison to literature results Table 7.2 shows the comparison of the results on effort from the CCC experiment with the Arisholm 2007 Experiment [3].

The table is grouped into the three experience levels and an overall perspective. Each group is split into simple and complex tasks and a row that combines the two kinds.

Across all experience levels pairs needed more effort in the CCC- compared to the Arisholm-experiment.

Juniors achieved a better result in simple tasks in at the CCC but a worse for complex tasks. This result contradicts to the findings in the Lui and Chan experiment [35], because they stated that PP does pay off when the problem is not known (complex). However, there is a risk that the values in table 7.2 are not accurate enough for juniors because only six juniors teams (two singles and four pairs) completed level three.

Intermediates competed far worse at the CCC than in the Arisholm experiment. According to Arisholm et al., intermediates should profit the most from PP but the CCC results are contradicting to this finding.

Exerience level	Complexity	CCC Experiment	Arisholm 2007 Experiment
Overall	All	124%	84%
	Simple	115%	60%
	Complex	142%	112%
Juniors	All	177%	111%
	Simple	93%	109%
	Complex	344%	112%
Intermediates	All	131%	43%
	Simple	137%	22%
	Complex	118%	68%
Professionals	All	80%	83%
	Simple	80%	55%
	Complex	82%	115%

Table 7.2: Comparison of this thesis' experiment and the Arisholm 2007 [3] experiments results about the effects of programming in pairs regarding effort. The numbers show the increase and decrease of time team needed when using pair-programming.

The CCC results for professionals fit well to the results from Arisholm. They only vary in the simply and complex subgroups showing that professionals competed as good at simple and complex tasks regarding duration and effort in the CCC experiment.

Discussion Due the almost equal results in duration the effects on effort favors the singles because the duration of pairs is doubled. The results of the T-tests confirm this logic: Across all experience levels the significance number amounts 0.000 for the first three levels of the competition and similar numbers when zooming into each experience group.

The null hypothesis 3.2.0 (No difference between pairs and singles) can be rejected but the expected alternative hypothesis 3.2.1 (Pairs need less effort than individuals) not accepted because the opposing effect occurred. Pairs need significantly more effort to complete programming tasks than individuals.

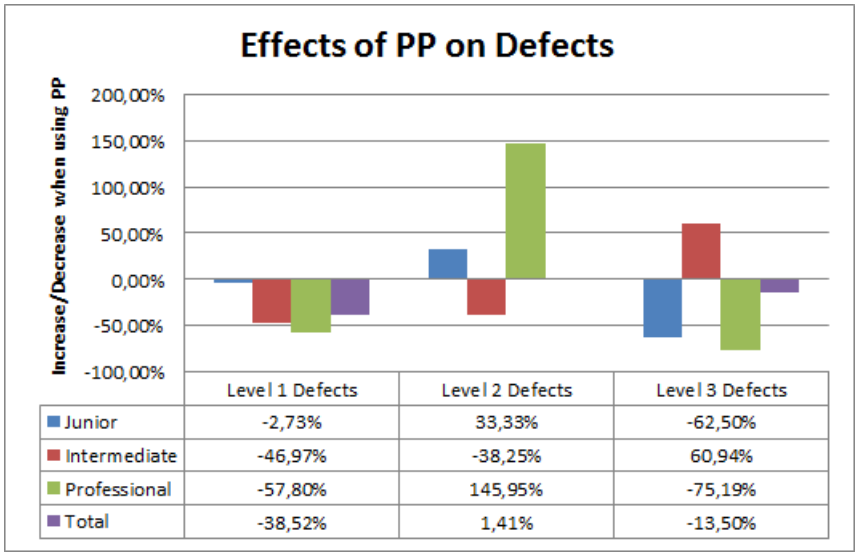
Effects of Pair-Programming on programming efficiency regarding defects Figure 7.4 shows the effect of PP on defects made in the first three levels, grouped by the experience level of the participants.

In level one, junior pairs made 3 % less defects than junior individuals. Intermediates made 47 % less and professionals 58 % less.

In level two, junior pairs made 33 % more defects than individuals. Intermediates 38 % less and professionals 146 % more defects.

In level three, juniors made 62 % less, intermediates 61 % more and professionals 75 % less defects.

In total, juniors and professionals made fewer defects when using Pair-Programming. Intermediates didn't profit from it, neither competed worse.



	Level 1 Defects	Level 2 Defects	Level 3 Defects
Junior	-2,73%	33,33%	-62,50%
Intermediate	-46,97%	-38,25%	60,94%
Professional	-57,80%	145,95%	-75,19%
Total	-38,52%	1,41%	-13,50%

Figure 7.4: Bar chart that shows the percentage difference between pair-programming teams and individual-programmers regarding the defects made. A positive number means, that pairs made more defects than individuals. Juniors and professionals profited from using PP.

Comparison to literature results Figure 7.3 shows the results of the CCC and Arisholm [3] regarding defects. Note, that the values of Arisholm have been inverted because they used correctness instead of a defects variable. Like in the previous comparisons, the first two levels are considered as simple tasks, while level three is considered as complex. In the table the values for "simple" are the mean of the first two levels, while the value for "complex" is the value of level three.

Regarding defects, the CCC shows an overall better result than the Arisholm experiment. Pairs made 16,9 % less defects than individuals (in the first three levels). For simpler tasks this result increased to 18,6 % while more complex tasks only had an advantage of 13,5 %. The overall results fits to Arisholm but the simple and complex are contradicting. At Arisholm, PP achieved an improvement of defects on complex tasks but an decline for simple tasks. At the CCC the results of both complexities were both better for pairs than for individuals.

In Arisholm et al.' experiment, juniors benefited the most from PP. This is approved by the CCC, however, the results were not as advantageous.

Intermediates benefited as lot from PP in simple tasks since they made 42,6 % less defects. On the contrary, intermediate pairs failed at complex tasks since they made 60,9 % more defects than individuals. This finding contradicts do Arisholm because there the intermediate pairs had 92% less defects.

Professionals had a similar result in the CCC like at the Arisholm experiment but the result was more spread. Further, professionals pairs made 44,1 % more defects on simple tasks while they made -75,1 % less on complex ones.

Exerience level	Complexity	CCC Experiment	Arisholm 2007 Experiment
Overall	All	-17%	-7%
	Simple	-19%	16%
	Complex	-14%	-48%
Juniors	All	-11%	-73%
	Simple	15%	-32%
	Complex	-63%	-149%
Intermediates	All	-8%	-4%
	Simple	-43%	29%
	Complex	61%	-92%
Professionals	All	4%	8%
	Simple	44%	13%
	Complex	-75%	2%

Table 7.3: Comparison of this thesis' experiment and the Arisholm 2007 [3] experiments results about the effects of programming in pairs regarding defects. The numbers show the increase and decrease of defects made when using pair-programming.

Discussion The results about the effects of Pair-Programming on defects made showed now significant results. Across all experience levels the results didn't show meaningful differences except for one slightly-significant case (0.095) in level five, where pairs were better than individuals.

The null hypothesis 3.3.0 (No difference between pairs and singles) cannot be rejected.

7.2.2 What are the effects of experience on coding efficiency in the context of programming contests? (RI4)

Effects of experience on programming efficiency regarding duration Figure 7.5 shows a summary of the effects of experience on duration. A positive number shows that more experienced teams competed worse than low experienced teams, as well as the other way around. The figure is also split into three rows: The first one shows a comparison of single juniors against single professionals, the second one pair juniors against pair professionals and the third the total value of the first two rows. The three columns describe the first levels of the contest/experiment. The first two are considered to be more simple while the third level was complex.

Across all three levels the professionals competed better than the juniors in a total context. For single participants the results were even or slightly better for professionals. Pairing professionals constantly achieved their tasks faster than pairing juniors. Especially in the complex level three, the pairing professionals exceeded the pairing juniors.

Discussion In level three of the experiment, the professionals worked faster than juniors with a significance value of 0.030. This effect didn't occur when looking at individual programmers, but the significance number of professional pairs working faster than junior-pairs amounts 0.023.

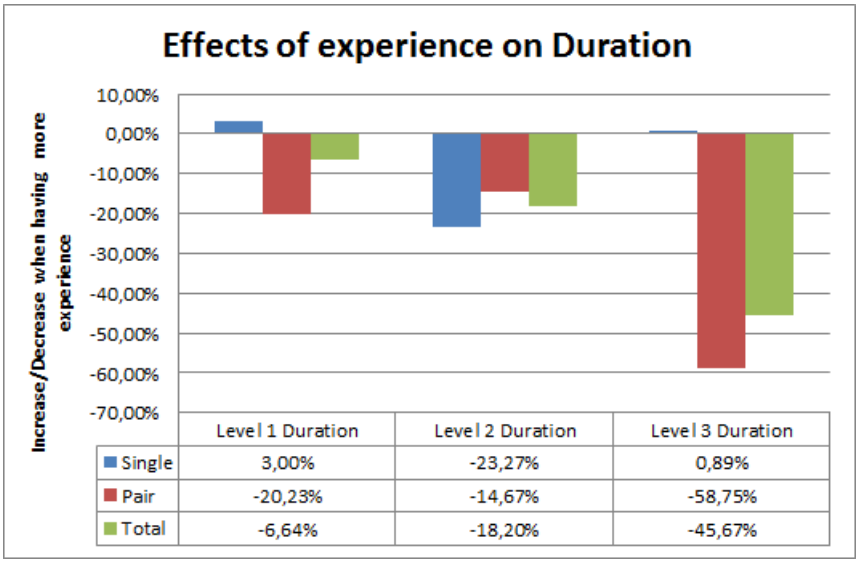


Figure 7.5: Shows the comparison of single juniors against single professionals, pairing juniors against pairing professionals and the total values for the first three levels of the CCC experiment regarding time needed to complete a task. A positive number represents an increase of time needed, when using pair-programming.

Therefore the null hypothesis 4.1.0 (Professionals and juniors work equally fast) can be rejected for complex problems (since level three was considered to be complex.) The alternative hypothesis (Professionals work faster than juniors) can be accepted with the restriction to complex problems.

Effects of experience on programming efficiency regarding defects Figure 7.6 shows the summary of the effects of experience on defects. The structure of the figure is the same like in figure 7.5.

The results show, that professionals made fewer defects than juniors across all levels and team sizes. The only exception is in level two where senior pairs made more defects than junior pairs.

Discussion There were no significant or slightly-significant values when the number of defects produced by professionals and juniors.

Therefore the null hypothesis 4.2.0 (Professionals and juniors produce the same amount of defects) cannot be rejected.

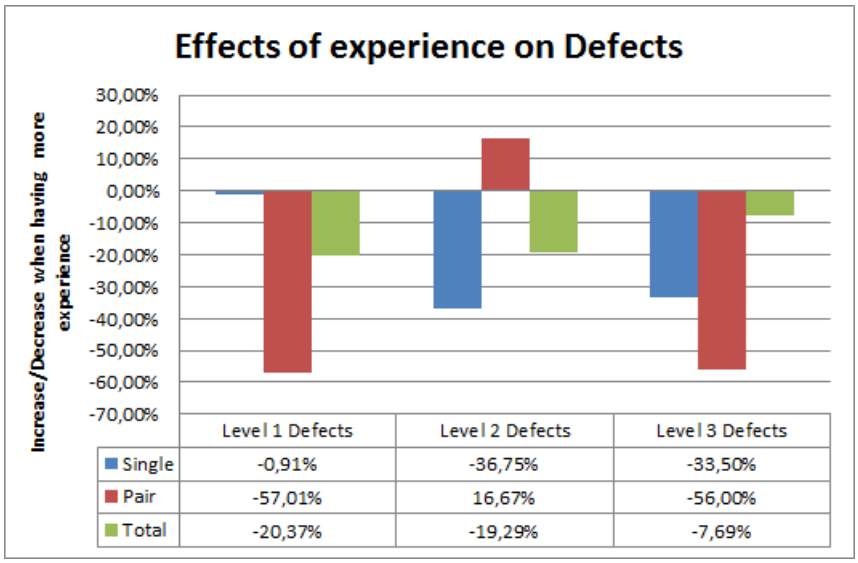


Figure 7.6: Shows the comparison of individual juniors against individual professionals, pairing juniors against pairing professionals and the total values for the first three levels of the CCC experiment regarding defects made. A positive number represents an increase of defects made when using pair-programming.

CHAPTER 8

Conclusion

We presented coding contests as a tool to conduct empirical studies and experiments in the field of software engineering. As a proof of concept, an experiment was embedded into a public coding contest and an empirical study was created based on the results.

We discussed four research issues, contained in two research topics (RT): the first two handled the embedding of an experiment into coding contests and how it can benefit scientific research. The later two issues were for discussing the results of the experiment by evaluating the effects of Pair-Programming and programming experience regarding duration, effort and defects.

In this chapter we summarize the methodological approach, the results and the interpretations and suggest future work.

8.1 Methodological approach

We planned and executed an experiment according to the approach specified in chapter 4.

The approach was 1) planning and executing of the experiment, 2) analysis of the data and 3) interpretation.

1) Planning and execution The experiment was planned and executed applying the process suggested by Wohlin's [64]. This phase consisted of a definition and planning phase where the goals for the experiment were defined and all material prepared. The second step was to observe the execution and ensure the data collection was working properly. For this step we were present at the coding contest located in Vienna. Lastly, the data was extracted from the Catalysts tools, validated and transformed into a structure that fits further analysis.

2) Data analysis In this step we cleaned up the data, e.g. removed invalid data points and reconstructed missing values, and prepared the data for later hypotheses testing. I.e. reduce the data set to the groups under investigation and use descriptive analytics tools to aggregate values

by which the groups can be compared. We also used analysis of variance tools like the t-test to evaluate the significance of the results.

3) Interpretation In the last step, the data was interpreted and the result for the research issues were discussed.

8.2 Participants of the experiment

In the experiment 173 teams completed at least one task. The team sizes were: 55 % individuals, 35 % pairs and 10 % teams of three. By experience the participants were grouped into: 23 % juniors, 53% intermediates and 24 % professionals.

The distribution of team size and experience fitted our expectations and provided enough material to work with.

We have been unsatisfied with he validity of the data: Some data points were missing (around 16 %) and thus, their results were excluded from further analysis. The reason was incorrect or incomplete registration data by the participants.

8.3 Research issues and results

This section summarizes the two research topics including their four research issues.

8.3.1 Coding contests as vehicle for empirical research (Research Topic 1)

- **How can coding contests support controlled experiments? (RI1)** The focus of this research issue lay on how coding contests can be used for embedded experiments in general. We discussed the experiences and lessons learned from the conduction of our experiment (Catalysts Coding Contest - Vienna 2012) and presented the results.

  The experiment was executed successfully and it proved enough material to test the hypotheses in RT 2. The expectations that researchers can save time and budget were fulfilled. The organizing company (Catalysts GmbH) planned and executed the coding contest, while we provided the tasks for the participants. During the execution we were locally present at the contest and observed the experiment. Shortly after the contest (less than a day) we received the data from Catalysts and could start with the extraction and preparation. Most of the data was spreadsheet-based so it was easy to migrate it into SPSS, our chosen statistic tool.

  The level based design was the key feature why the conduction worked so well. It gives pair programmers and individuals equal chances because each levels' problem was single-layered and every team can only work at one level at a time. Thus, pairs cannot solve different problems in parallel. The level based design also lets us control the order in which the tasks have to be completed.

  Our tasks for the experiment were split into simple and complex problems. The first two levels were simple and the later ones complex. However, the time limit of the experiment

was too short for everyone to finish all tasks. (This is a requirement by Catalysts to give everyone a challenge). Thus, many participants completed only the simple tasks but no complex ones. This is unfortunate, because it limits the usage of the collected data.

A drawback of our experiment was that the skill level of the participants differed drastically. At least two former IOI end round participants attended at our experiment. Due to their young age they were counted as junior programmers but their actual skill level lies higher. This can lead to distortions in the results. More information about the skill level of each participant would be necessary to prevent this risk (E.g. a survey before or after the contest for each participant that allows a more neutral rating of each ones skill level).

- **How can the concept of coding contests support research and replication in the field of empirical software engineering? (RI2)** This research issue handled the benefits for empirical research when embedding experiments into coding contests. In contrast to RI1, the discussion was more generally discussed and not focused on a coding contest.

  Our first finding is, that our concept supports systematically repeatable experiments. The Catalysts Coding Contest (CCC) is held thrice per year and is planned to do so in the future. Thus, a researcher can easily create replications of former experiments without much effort within a year. It supports systematical replication because a researcher can try out the effects of one or more variables with every contest.

  The participants (see section 8.2 above) formed a heterogeneous group consisting of juniors, intermediates and professionals teamed up as individuals, pairs or teams of three. The youngest ones were high school students around the age of 16 and the oldest ones were experienced programmers in their 40s and 50s. The main group formed students and young professionals. The high number of participants allows researchers to filter them to smaller groups and still have enough data points that allow empirical scientific work.

  Other CCCs have been conducted during the time this thesis was written. We observed their executions but did not analyze their results in detail (see section 8.4 future work). However, the rules of the contest have not been changed since our experiment and the execution followed the same approach. This substantiates our finding, that the CCC allows easy replication.

  We suggest the use of common data models (See Biffl et al. [11]) so that different researchers, who conducted experiments in other coding contests, can share their results easier.

8.3.2 Empirical evaluation of software-development-techniques effects in context of coding contests (Research Topic 2)

- **What are the effects of developing software in teams? (RI3)** In RI3, we examined the results of the experiments and compared individual programmers against pair programmers. In three hypotheses we tested differences in duration, effort (person hours) and defects made. RT 2 is acting as a proof of concept for RT 1. We expected that Pair-Programming will achieve better results.

Our findings were:

Regarding duration needed to complete tasks: Professional pairs are slightly faster than professional individuals. Pairing juniors and intermediates needed slightly more time to complete tasks than individual juniors and intermediates. For complex problems, the time needed was much higher for pairing juniors than for individual juniors. However, none of these findings are significant.

Regarding effort needed to complete tasks (person hours): The general effort doubled for pair programmers (significantly). This result was already predictable from the results of the duration part. Since pairs and individuals needed the same time to complete tasks, the person hours needed by pairs will double.

Regarding defects produced by tasks: The findings are mixed and vary for each task. Depending on the task and skill level, the results varied, meaning that sometimes the individuals produced fewer defects and sometimes the pairs. Generally, there seem to be a benefit for pairs, but not significantly.

- **What are the effects of experience on coding efficiency in the context of programming contests? (RI4)** RI4 compares the results from junior-programmers and professionals regarding the duration needed to complete a task and defects made. We expected that professionals compete better than juniors.

  As expected, the professionals competed better than the juniors but the results are not significant.

8.4 Future work

In our experiment we focused on skill level and team size and analyzed duration, effort and the number of defects made. However, the data collected during the execution of the experiment provides more data points than we looked at:

Group teams by number of completed levels In chapter 6 we presented the data of RI3, further grouped by the number of levels completed by each team. This approach seems to provide more meaningful results: in one group all pairs were significantly faster or made fewer defects than individuals. Another study could examine these results further or replicate the experiment to check if this approach is more useful.
See 6.3.1, 6.3.2 and 6.3.3 for the statistics.

Type of defects Instead of only looking at the total numbers of defects made in each task, a researcher can also analyze each defect individually. Catalysts archives the complete contest log files, so the result of each submitted test case is stored. I.e. we know the exact values the participants submitted for each test case. This allows us to test if a participant made particular logical mistakes.
However, this means a lot of effort for the researcher.

Other CCCs During the time this thesis was written, other CCCs were executed and their results barely evaluated. A researcher can make a meta study by asking for the past CCCs data and comparing the results of each CCC with each other.

Coding language A researcher can analyze the effects of different coding languages on the results of the participants.

Source code analysis Because the participants upload their source code (to obtain a time bonus), a researcher can analyze their source files for various reasons.
E.g.: Due to the level based design, we can analyze how the code evolved after each level and check if somebody uses clever refactoring methods. Or, one can check if a participant used test driven development.

Adaptations of CCC A researcher should also think about adaptations that can be made to the CCC. E.g. more time for the participants. Or, special tasks (levels) that test how well a participant possesses various programming skills (Can they handle huge amounts of data? Do they know efficient algorithms for a problem?)

Focus on professional programmers Future studies should focus on using coding contest to measure the effects of SW development methodologies on professional programmers. Thus, these studies would be more meaningful for drawing general assumptions.

Bibliography

[1] Saman Amirpour Amraii. Observations on teamwork strategies in the ACM international collegiate programming contest. *Crossroads*, 14(1):1–9, December 2007.

[2] Steven K Andrianoff, Dalton R Hunkins, and David B Levine. Adding Objects to the Traditional ACM Programming Contest. In *Proceedings of the twenty-fourth SIGCSE technical symposium on Computer science education (SIGCSE '04)*, pages 105–109, 2004.

[3] Erik Arisholm, Hans Gallis, Tore Dybå, and Dag I.K. Sjø berg. Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Transactions On Software Engineering*, 33(2):65–86, 2007.

[4] Owen Astrachan, Vivek Khera, and David Kotz. The internet programming contest: a report and philosophy. In *Proceedings of the twenty-fourth SIGCSE technical symposium on Computer science education (SIGCSE '93)*, pages 48–52, 1993.

[5] V. R. Basili. The Experimental Paradigm in Software Egineering. In H.D. Rombach, V. R. Basili, and R.W. Selby, editors, *Experimental Software Engineering Ussues: Critical Assessment and Future Directives*. Springer-Verlag, 1993.

[6] Victor R Basili. Quantitative evaluation of software engineering methodology. In *Proceedings of the First Pan Pacific Computer Conference*, pages 379–398, 1985.

[7] Victor R Basili, Forrest Shull, and Filippo Lanubile. Building Knowledge through Families of Experiments. *IEEE Transactions On Software Engineering*, 25(4):456–473, 1999.

[8] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1999.

[9] Kent Beck. *Test-driven development: by example*. Addison-Wesley, 2003.

[10] Andrew Begel and Nachiappan Nagappan. Pair Programming : What's in it for Me? In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM'08)*, pages 120–128, 2008.

[11] Stefan Biffl, Estefanía Serral, Dietmar Winkler, Nelly Condori-Fernández, Oscar Dieste, and Natalia Juristo. Replication Data Management: Needs and Solutions – An Initial Evaluation of Conceptual Approaches for Integrating Heterogeneous Replication Study

Data. *Empirical Software Engineering and Measurement (ESEM'2013)*, pages 233,242, 2013.

[12] Tanja Bipp, Andreas Lepper, and Doris Schmedding. Pair programming in software development teams – An empirical study of its benefits. *Information and Software Technology*, 50(3):231–240, February 2008.

[13] Raewyn Boersen and M. Phillipps. Programming contests: Two innovative models from New Zealand, 2006.

[14] Grant Braught, John MacCormick, and Tim Wahls. The Benefits of Pairing by Ability. In *Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10)*, pages 249–253, New York, New York, USA, 2010. ACM Press.

[15] Alistair Cockburn and Laurie Williams. The Costs and Benefits of Pair Programming. *Extreme programming examined*, 96(1123):1–11, 2001.

[16] Larry L. Constantine. *Constantine on Peopleware*. Prentice Hall Ptr, 1995.

[17] T. D. Cook and D. T. Campbell. *Quasi-Experimentation – Design and Analysis Issues for Field Settings*. Houghton Mifflin Company, 1979.

[18] Gordon Cormack, Ian Munro, Troy Vasiga, and Graeme Kemkes. Structure , Scoring and Purpose of Computing Competitions. *Informatics in Education*, 5(1):15–36, 2006.

[19] Enrico Di Bella, Ilenia Fronza, Nattakarn Phaphoom, Alberto Sillitti, Giancarlo Succi, and Jelena Vlasenko. Pair Programming and Software Defects - A Large, Industrial Case Study. *IEEE Transactions on Software Engineering*, PP(99):1–31, 2012.

[20] Torgeir Dingsø yr, Sridhar Nerur, VenuGopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6):1213–1221, June 2012.

[21] Sebastian Dyck and Tim A. Majchrzak. Identifying Common Characteristics in Fundamental, Integrated, and Agile Software Development Methodologies. In *45th Hawaii International Conference on System Sciences*, pages 5299–5308. Ieee, January 2012.

[22] Fabian Ernst, Jeroen Moelands, and Seppo Pieterse. Programming contest strategies. *Crossroads*, pages 17 – 19, 1996.

[23] Ilenia Fronza, Alberto Sillitti, and Giancarlo Succi. An Interpretation of the Results of the Analysis of Pair Programming during Novices Integration in a Team. *3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, pages 225–235, 2009.

[24] Robert L Glass. The Software Research Crisis. *IEEE Software*, November:42–47, 1994.

[25] Boris Gloger. Scrum. *Informatik-Spektrum*, 33(2):195–200, March 2010.

[26] Brian Hanks. Student Attitudes toward Pair Programming. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education (ITICSE '06)*, pages 113–117, 2006.

[27] Brian Hanks. Empirical evaluation of distributed pair programming. *International Journal of Human-Computer Studies*, 66(7):530–544, July 2008.

[28] Jo E Hannay, Erik Arisholm, Harald Engvik, and Dag I K Sjø berg. Effects of Personality on Pair Programming. *IEEE Transactions on Software Engineering*, 36(1):61–80, 2010.

[29] Jo E. Hannay, Tore Dybå, Erik Arisholm, and Dag I.K. Sjø berg. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7):1110–1122, July 2009.

[30] Shokoofeh Hesari, Hoda Mashayekhi, and Raman Ramsin. Towards a General Framework for Evaluating Software Development Methodologies. In *34th Annual IEEE Computer Software and Applications Conference*, pages 208–217. Ieee, July 2010.

[31] H. Hulkko and P. Abrahamsson. A Multiple Case Study on the Impact of Pair Programming on Product Quality. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 495–504. IEEe, 2005.

[32] Andreas Jedlitschka and Markus Nick. Software Engineering Knowledge Repositories. *Empirical Methods and Studies in Software Engineering*, 2765:55–80, 2003.

[33] Natalia Juristo and Ana M Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Boston, 2001.

[34] Andreas Lepper. *Pair Programming - Methodiken und Ergebnisse einer Studie (German)*. VDM Verlag Dr. Müller, 2007.

[35] Kim Man Lui and Keith C.C. Chan. Pair programming productivity: Novice–novice vs. expert–expert. *International Journal of Human-Computer Studies*, 64(9):915–925, September 2006.

[36] Kim Man Lui, Keith C.C. Chan, and John T. Nosek. The Effect of Pairs in Program Design Tasks. *IEEE Transactions on Software Engineering*, 34(2):197–211, March 2008.

[37] Konstantinos Manikas and Klaus Marius Hansen. Software ecosystems – A systematic literature review. *Journal of Systems and Software*, 86(5):1294–1306, May 2013.

[38] Shahriar Manzoor. Common mistakes in online and real-time contests ALT. *Crossroads, The ACM Student Magazine*, 7(5):10–16, July 2001.

[39] Shahriar Manzoor. Analyzing Programming Contest Statistics, 2006.

[40] Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. The effects of pair-programming on performance in an introductory programming course. *Proceedings of the 33rd SIGCSE technical symposium on Computer science education (SIGCSE '02)*, 34(1):38–42, 2002.

[41] Charlie Mcdowell, Linda Werner, Heather E. Bullock, and Julian Fernald. Pair Programming Improves Student Retention, Confidence And Program Quality. *Communications of the ACM*, 49(8):90–95, 2006.

[42] Matthias M. Müller. Two controlled experiments concerning the comparison of pair programming to peer review. *Journal of Systems and Software*, 78(2):166–179, November 2005.

[43] Matthias M. Müller and Frank Padberg. An Empirical Study about the Feelgood Factor in Pair Programming. *Proceedings of the 10th International Symposium on Software Metrics (METRICS'04)*, 2004.

[44] Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. Improving the CS1 Experience with Pair Programming. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE '03)*, pages 359–362, 2003.

[45] Nachiappan Nagappan, Laurie Williams, Eric Wiebe, Carol Miller, Suzanne Balik, Miriam Ferzli, and Julie Petlick. Pair Learning : With an Eye Toward Future Success. In *Extreme Programming/Agile Universe*, pages 185–198, 2003.

[46] Jerzy Nawrocki and Adam Wojciechowski. Experimental Evaluation of Pair Programming. In *European Software Control and Metrics (Escom'01)*, pages 99–101, 2001.

[47] John T. Nosek. The Case for Collaborative Programming. *Communications of the ACM*, 41(3):105–108, March 1998.

[48] Frank Padberg and Matthias M. Müller. Analyzing the Cost and Benefit of Pair Programming. In *Proceedings of the Ninth International Software Metrics Symposium (METRICS'03)*, pages 166–177, 2003.

[49] Allen Parrish, Randy Smith, David Hale, and Joanne Hale. A Field Study of Developer Pairs: Productivity Impacts and Implications. *IEEE Software*, 21(5):76–79, 2004.

[50] David a. Patterson. Reflections on a programming Olympiad. *Communications of the ACM*, 48(7):15–16, July 2005.

[51] Mauro Pezze and Michal Young. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley & Sons, 2007.

[52] Nattakarn Phaphoom, Alberto Sillitti, and Giancarlo Succi. Pair Programming and Software Defects – An Industrial Case Study. *Lecture Notes in Business Information Processing (XP'2011)*, pages 208–222, 2011.

[53] Monvorath Phongpaibul and Barry Boehm. An Empirical Comparison Between Pair Development and Software Inspection in Thailand. In *Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering - ISESE '06*, pages 85–94, New York, New York, USA, 2006. ACM Press.

[54] Bill Poucher. Giving students the competitive edge. *Communications of the ACM*, 55(8):5–5, August 2012.

[55] Colin Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. Blackwell Publishers, 2002.

[56] Norsaremah Salleh, Emilia Mendes, and John C. Grundy. Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 37(4):509–525, July 2011.

[57] Nikolay V. Shilov and Kwangkeun Yi. Engaging students with theory through ACM collegiate programming contest. *Communications of the ACM*, 45(9):98–101, 2002.

[58] Irfan Naufal Umar and Tie Hui Hui. Learning Style, Metaphor and Pair Programming: Do they Influence Performance? *Procedia - Social and Behavioral Sciences*, 46:5603–5609, January 2012.

[59] Laurie Williams. Integrating pair programming into a software development process. In *Proceedings 14th Conference on Software Engineering Education and Training. 'In search of a software engineering profession' (Cat. No.PR01059)*, pages 27–36. IEEE Comput. Soc, 2001.

[60] Laurie Williams and Robert R Kessler. *Pair Programming Illuminated*. Addison-Wesley, 2002.

[61] Laurie Williams, Robert R Kessler, Ward Cunningham, and Ron Jeffries. Strengthening the Case for Pair Programming. *IEEE Software*, 17(4):19–25, 2000.

[62] Laurie Williams, Charlie McDowell, Nachiappan Nagappan, Julian Fernald, and Linda Werner. Building Pair Programming Knowledge through a Family of Experiments. In *Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)*, pages 143–152, 2003.

[63] Dietmar Winkler, Martin Kitzler, Christoph Steindl, and Stefan Biffl. Investigating the Impact of Experience and Solo/Pair Programming on Coding Efficiency: Results and Experiences from Coding Contests. In *Proceedings of the 14th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP'2013)*, pages 106–120, 2013.

[64] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Introduction to Experimentation in Software Engineering*. Springer, 2000.

[65] Stephen Wood, George Michaelides, and Chris Thomson. Successful extreme programming: Fidelity to the methodology or good teamworking? *Information and Software Technology*, 55(4):660–672, April 2013.

[66] Stuart Wray. How Pair Programming Really Works. *IEEE Software*, pages 50–55, 2010.

[67] Shaochun Xu and Vaclav Rajlich. Empirical Validation of Test-Driven Pair Programming in Game Development. *5th IEEE/ACIS Int. Conf. on Comput. and Info. Sci., ICIS 2006*, pages 500–505, 2006.