

**Learning Processes and Consistency in the Quality of Software
Development Based on Environmental Influences**

by

Michael Christian Frick, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2016

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Michael Christian Frick

July 26, 2016

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Michael Christian Frick

July 26, 2016

Acknowledgments

Give many thanks and stuff...

MICHAEL CHRISTIAN FRICK

University of Dublin, Trinity College

September 2016

**Learning Processes and Consistency in the Quality of Software
Development Based on Environmental Influences**

Publication No. _____

Michael Christian Frick, M.Sc.
University of Dublin, Trinity College, 2016

Supervisor: Stephen Barrett

Abstract

this is my abstract

Contents

Acknowledgments	iv
Abstract	v
List of Tables	x
List of Figures	xi
Chapter 1 Introduction	1
1.1 Motivation	4
1.1.1 Mobile Device Sensors	4
1.1.2 Learning to write code	4
1.1.3 The importance of Software Metrics	5
1.1.4 Working environment	6
1.2 Aims	7
1.3 Road-map	7
Chapter 2 The State of the Art	8
2.1 Software Metrics	9
2.2 Metrics Measurement and Analysis	10
2.3 Mobile Data Gathering	12
2.4 Data Clustering	13
2.5 Variable Quality Influences	15

2.6	Team Communication	15
2.7	Cognitive Performance	17
2.7.1	Context Switching	17
2.7.2	Working Environment	18
2.7.3	Arousal Effects	19
2.7.4	Diet	20
Chapter 3 Design		21
3.1	Functionality Overview	21
3.2	Mobile Application Design	22
3.2.1	App Architecture	23
3.2.2	User Interface	24
3.2.3	Data Storage	24
3.2.4	API	25
3.2.5	User Information	26
3.3	Server Design	27
3.3.1	Server Frontend Design	27
3.3.2	Server Backend Design	27
3.4	Tools	27
3.4.1	Decrypting Tool	28
3.4.2	Data Structuring Tools	28
Chapter 4 Implementation		29
4.1	Android	29
4.1.1	User Interface	30
4.1.2	Data Gathering	32
4.1.3	Data Storage	33
4.1.4	Security	33
4.2	Server	35

4.2.1	Back-End	35
4.2.2	Webpages	35
4.3	Tools	36
4.3.1	Encryption Tool	36
4.3.2	User Separation Tool	36
4.3.3	Value Separation Tool	36
4.3.4	Latex Plot Syntax Creating Tool	37
Chapter 5 Experiment		38
5.1	Crowd Experiment	38
5.1.1	Setup and Execution	38
5.1.2	Classification	40
5.1.3	Questions	44
5.2	Individual Experiment	45
5.2.1	Setup and Execution	45
5.2.2	Scenarios	46
Chapter 6 Results		48
6.1	Individual Experiment Results	49
Chapter 7 Conclusions		51
7.1	Future Work	51
Appendix A Abbreviations		52
Appendix B Source Code		54
Bibliography		55

List of Tables

5.1	Common Outdoor Light Levels	40
5.2	Common & Recommended Indoor Light Levels	41
6.1	Cognitive Performance with Coffee	49

List of Figures

3.1	Smartphone OS Marketshare	22
3.2	Android Views	24
4.1	Security Dataflow	34
5.1	Experiment Execution	39
5.2	Android Views	42

Chapter 1

Introduction

"Be a yardstick of quality. Some people aren't used to an environment where excellence is expected."

Steve Jobs

A computer is the powerful hardware that is able to do much more than a human being is capable of. However, The enormous capacity is useless without software which is making use of its power. In order to make the increasing computations controllable and keep the machines as customizable as possible, different layers of abstraction allow to make the computing power manageable. The abstraction reaches from machine readable code over Assembly to high level programming languages which can almost be read as an English sentence. On top of that, there are libraries and frameworks that already provide features that already have been implemented [30]. A big part of the complexity is already encapsulated and the software engineers can focus on the functionality of the software and their specific problems.

Over the years, the performance of the computers rapidly increased and with it the

complexity of the code [45]. Software can be a simple tool that is written in a short time by a single person or it can be a gigantic software project with several hundred developers working on it[10]. In order to allow to split the work on a software project, an encapsulation of the modules is mandatory. A general structure must be given to ensure that the different parts can integrate hand in hand and to keep the code understandable. The more people work on one project, the more important it is to provide a organized, well planned architecture and keep the code clean.

In today's world, software is everywhere. The traffic is controlled by computers, security systems, nuclear power plants or just a messenger app on a mobile phone etc. The ubiquity of computers can make life easier, but can also cause unpredictable trouble.

In the early 1890s at the United Kingdom's Royal Air Force an engineer found a bug that could have fired a missile without any command. Luckily it was found before a disaster happened [38]. The quality requirements varies for different software products. A crashing weather app on a mobile is not as bad a bug causing a production stop in a plant. However, the quality of the software can make the difference whether a company will be successful or just be one of many abortive start-ups with a good idea but a bad execution.

In the software industry, the most significant factor in the creating process is the human. The quality strongly depends on the performance of the programmers as a single person or in a team. That performance quality can change by various different reasons even a few times a day.

Previous researchers already did a lot of work and research in that area. will start the dissertation, summarizing their findings which will include previous studies that investigated different theories about influencing factors on cognitive performance and related work.

In this dissertation, I will write about my approach where I used the sensors and informa-

tion provided by the user's mobile phone to find evidence in factors that influence software quality. I developed an Android application for being installed on the participants mobile device. The app is gathering the location of the user, collecting sensor data from the light sensor, accelerometer, the environmental noise from the microphone and the data from the step counter of the device.

The sensors and the software gather the data while the participant is solving a provided programming question. Afterwards the user gets asked to answer some questions such as information about disturbance or for example mood and more.

Based on that information I tried to find patterns in behavior and environment that are influencing the quality of a programmer. The gathered information were clustered into a specific classified behavior or context and I compared them to find correlations with the code quality.

For determining the code quality I used a tool that can analyze the code that is uploaded on GitHub and calculate a level of quality.

SOME RESULTS WILL BE MENTIONED

In the future, the app could become an every-day tool for developers and students. It could be used instead or hand in hand with project management tools, that require the exact times how long a programmer worked on a project. It could simultaneously provide real time feedback about the code quality itself or suboptimal aspects in the working environment. Rather than comparing the information with other app users, the app could make use of systematic learning of and optimal working environment and behavior of the specific programmer. Having this information, the app could inform the user when the optimal environment is not given and then provide the information how it could be reached (e.g. "Find a place with better lighting conditions and less environmental noise to increase your productivity").

1.1 Motivation

1.1.1 Mobile Device Sensors

Over the last decades the evolution mobile devices began with a wireless telephone far away from pocket size. Over the years the mobile devices got displays, SMS, telephone books, games and a lot more. In 2007, Steve Jobs introduced the first iPhone and with it the age of the smartphone [28]. Over the years, smart-phones became pocket size computers with a better display resolution than the most televisions and the computing power of what desktop pc users could just dream about a few years ago. More and more sensors were packet into the small handy devices were made easy accessible by developers. The sensors range from proximity detection over accelerometer to humidity sensors etc. Google even engineered a system for 3D objects and indoor environments with just a single device in real time [40].

So, a lot of people own the hardware with the capability to collect rich context information and they even carry it with them all the time and could be used for support and improve the people's work and environment.

1.1.2 Learning to write code

Learning how to program is getting more and more important and still not a required subject in school. It is the computer with its software which is controlling almost everything in our everyday life such as traffic, gates, calendars etc. I failure could have dramatic impacts in peoples life.

Thus, it is very important that programmers produce high quality code and also be able to find good frameworks and libraries. The problem is that it is not always obvious what high quality means. It can vary from good structured code to resource-aware, reliability and much more. A lot of programmers didn't learn coding in school or university. They

taught it themselves and they might just used it for fun-project which were not created for public usage. However, what I want to say is that a programmer might not really know how good or bad his/her code really is.

I experienced this problem myself. I started to work in an agency that specialized on iPad apps and design. I was hired because the previous mobile developers left the company and they urgently needed a replacement. When I started I had no practical experience in writing mobile applications. I had to maintain the current code and add new features in a big and unknown project. As I had no mentor or anyone who could give me feedback I just did it as good as I could. I still don't know whether I created good or bad code. With a feedback tool for my code quality I could have learned a lot about the coding itself and by , I would probably write much better code.

1.1.3 The importance of Software Metrics

Software is becoming more complex than ever and used in almost every environment. The deadlines in professional software projects are very strict and there is no time to develop everything from scratch. Development teams depend on libraries rather than reinventing the wheel over and over again.

The problem is that it's so easy to start programming and learning everything online from various sources. A lot of people do programming for a hobby and the quality and performance doesn't matter to them as much as in a professional environment. They also create libraries and frameworks with their quality standards. Also the increasing amount of open source libraries and the dependencies they create makes it very hard but also very important to professionals that they can trust these libraries and the quality. At this point of time the only indicators are user ratings and the amount of times it's been used in different frameworks and projects. Some frameworks are also recommended in public reviews articles.

Frameworks and libraries also need to be more dynamic and maintained. Operating

systems and programming languages are being updated more frequently which requires fast changes. A constant measuring of quality could at the first place give direct feedback that the quality stays constant after changes and as well helps the developer to create a better structure and code to improve the maintainability at the first place.

The most used platform in the open source community is Github. Github is based on git and is a web server that can be used to host software projects and allow to make them accessible to others developers [11].

1.1.4 Working environment

The feedback of the code quality can then be used to find and improve influencing factors. How important is a good working environment?

A new trend, especially in the tech industry is going from common clean looking office spaces to colorful creative environments closer to living rooms. Companies like Google or Facebook seem to get rid of the strict separation of work and personal life. Companies introduce unlimited holiday policies, provide free food and even have a laundry service for their employees. They try to remove all the obstacles from their employees life to allow them focus on their work. Also the social aspect at work changes a lot. Some years ago, things like having a beer with the co-workers after office hours or meeting the colleagues for a ping-pong match during the day was unthinkable.

Google tries to make developers communicate more with the team by placing the whole team in relatively small spaces and provide silent areas for tasks that require more silence. All these efforts to make the employees more productive are very interesting approaches but hard to measure.

In my dissertation I am trying to find patterns between working environment, behaviors and the resulting code quality in learning and professional environments. I also think that the creation of awareness for code quality and performance is an essential factor in the

evolution of a programmer and is important at any stage of the experience and my work could be a base for tools that are providing these information to the software developers and engineers.

1.2 Aims

I hope to find patterns in the working environment and behavior for in general that significantly influence the quality in software development. This knowledge might inspire and help future researchers and the industry to do more work in this area and create tools for bringing the code quality of future software to a higher standard. Also for academia, a tool that provides feedback on code quality for the students can help to bring them on a higher level when they leave college.

1.3 Road-map

In the next chapter I am summarizing the state of the art in the area of software metrics, its measurements and analysis, data gathering followed by data clustering and factors that influence cognitive performance and software quality.

Chapter 3 explains why it is the right time for this project and why it is important. The following Chapter describes the design of the different software components that are using to gather the data, provide information to the participants and to ensure the privacy of the gathered information. Chapter 5 contains information and the process of the implementation of the different software components. The experiment is being described in chapter 6 and includes the setup and execution, the expected results and the data classification as well as the questions that are being asked to the participants. The last two chapters describe and interpret the results and conclude the subject and information of the experiment

Chapter 2

The State of the Art

Many researchers are concerned about finding metrics of software quality with the human factor as the most significant part in the development process. Research has also been done in the process of finding and testing factors which are influencing the cognitive performance of software developers.

In my dissertation I aim to identify environmental influences that can improve, worsen or influence the code quality of Computer Science students and professional software developers. With the students as a key factor in future software engineering, I will analyze their behavior and performance influences in a environmental and psychological sense but also compare it with the numbers of professionals. [12]

Measuring quality of software projects and gaining information about the progress are valuable information for the software engineers and developers to reflect their performance. The provided feedback helps them to identify their weaknesses and improve their skills or optimize their work patterns [22] [30].

Also project managers have a great interest in details about the progress and the products quality in order to coordinate the schedules, resources and having an overview about the possible bottlenecks in the project. Early knowledge about potential problems can help

them to target it and make a difference between the success or failure of a software project.

A good programmer nowadays is described as a person who can solve complicated problems by breaking them down in smaller targeted problems that are easy to understand and to solve.

Good code is supposed to be clean, easy to read and as simplified as possible [22].

This new approach differs from the early days when programmers tried to find the shortest and most performant solutions. As long as code was using minimal resources it was fine. Less people were working on projects and the open source community was not as important and big as today. A lower computing power in that days made computers unable to handle the complexity that software has today.

More people are working together on different parts of complex systems. At the end every part must go hand in hand with all other parts and the code should have a similar structure so that people from one team could possibly also work or help out in another team.

From a research perspective it is very interesting to get an overview approaches from different years to gain a broader understanding. Thus the following paragraphs will summarize information about code metrics and code quality from several decades.

2.1 Software Metrics

Since the late 1960s, when the software engineering was in its beginning, people wanted to measure and produce numbers to characterize code properties. The first metrics were used and developed to measure and evaluate the performance of a programmer. Lines of Code (LOC) per month and bugs per thousand lines of code (KLOC) are a very simple but efficient ways for examining the productivity, which can be used to for comparison

with other programmers or general standards.

“Software Metrics“ is the term that has been used more then 30 years ago up to today. Today, some of the most metrics are still used to investigate the productivity of the software developers. The amount of bugs in relation to the amount of code, the initial number of requierements compared to the requierememts at the current point in the project and the effort it takes to fix faults versus the total time the project requieres. [25] The metrics have been a great success in the industry. Most of the big software companies and even smaller ones use metrics, though they are barely used in academia. The metrics are created for larger software and scopes. Also maintenance and re-factoring is not as important in academia as it is in the industry with commercial software. After all, Software metrics in the industry are primarily important for the management rather than for the development process.

Industrial software metrics can be used to ensure quality, productivity and can even make predictions of the software quality and it’s reliability [13].

Several researchers investigated and developed approaches to improve the metrics and the results which they are generating. Yue Jiang et. al. [21] from West Virginia University researched methods for improving software quality predictions. They used supervised machine learning algorithms with datasets and focused in improving of the information content of the training data in their research. The results at the end showed that the biggest differences in the quality of the predictions are generated by the choice of the right software metrics rather than applying different machine learning algorithm.

2.2 Metrics Measurement and Analysis

PSP - Personal Software Process is a way to gather data about the Software engineering process and analysis of the information. Over the last decades, the University of Hawaii did a lot of research in PSP and they developed different approaches to make students

to adopt and use it in their project and even later at work. Their first approach was originally described as “A Discipline for Software Engineering”. It required the users to keep records about all the metrics by hand. The massive overhead was a high barrier for the students to adopt and keep on working with the PSP. For the best results they needed to write down every compiler error and they had to track the time they were working on their projects and had to stop it for interruptions.

In 1998 the University of Hawaii started the Leap research project to provide a PSP with low overhead for the collection and analysis of the data. This generation of PSP was using automated tools which were asking the user for inputting the data. These tools were also able to display information and analyses to the user. Just a few students adopted the system. The researchers found out that another reason for the reluctant adoption was the constant context switches for the users. Inputting the data during the programming task interrupted and disturbed the ability to focus on the programming tasks. [24] In order to eliminate the adopting barriers, they started the Hackystat project in 2001. Hackystat is an open source framework for automatically gathering all the required metrics by data collection plugins in the development environments of the users.

Plugins, that are installed by the in their programming environments automatically collect the data and forward it to a centralized web service. The web service orders and analyzes the data. If interesting results occur, the webservice sends emails to the developers to inform them about it. The web service also provides a rich visual representation of the data. All the different approaches to provide feedback about the code lead to improvements in the quality and the ability to estimate software projects. [23] The Appalachian State University in North Carolina described a different approach. Their goal was to decrease the high attrition rate of computer science students and increase the attraction to get a computer science degree in general.

The researchers were monitoring the students software development behaviour in order to find good practises for successfully learning programming. For gathering the data of the individual students, they developed a tool called ClockIt. ClockIt allows to, fully auto-

matically, collect the data, analyze it and compare the results with the results from better or more experienced students visually. A web interface provides access to measurements for the student, the course instructor or an administrator.

In their results they compared the data of three students out of 75 participants. The students with the best results, an average scoring student and the one with the lowest grade. The comparison showed that the best student also spent the most time on the project, but wrote less code than the average scoring student who spend almost as much time. The worst student spend the least time and submitted the smallest amount of code. There was an interesting correlation between the grade and the compilation errors and the amount of compilations that were made. The best student compiled the code more than double as much as the average student and almost 6 times more than the worst student did. [32]

2.3 Mobile Data Gathering

Ferreira, D., et al. [14] from the University of Oulu, Finland and the Carnegie Mellon University were working on a toolkit for gathering the sensor data from mobile Android devices. They created an extensible framework that could have been used in any Android application at the time when the paper was released. They also released an application for research purposes (the Aware client). The Aware client is extendable with plugins to support more than the pre-installed sensors. By default, the application stores the gathered data on the local hard disk but can also be uploaded to a database.

The sensoring is optimized to keep the energy impact as little as possible and not to use more device resources than necessary.

Another approach in the area of mobile data gathering have been made by University of Science and Technology of China HUI XIONG, Rutgers University in cooperation with Nokia. In order to detect the context of the mobile device, Zhu, Hengshu, et al. [47] were

reading the log files of the device. The device logs provide information about location, accelerometers and optical sensor as well as browser history or which apps were used and are automatically recorded by the Android operating system. These information can be used to provide context aware suggestions e.g. for other games or based on the physical location. To read the logged information, the needs to be physically connected computer. The information from the device logs are much richer than the information which can be gathered in an application with the downside, that the device needs to be connected to a computer in order to access the information. A installed application can compute, store and transfer the data to a remote server from everywhere. The only requirement is access to the internet.

2.4 Data Clustering

In order to make give data a meaning it makes sense to cluster in a logical way and make assumptions about the needed features or contexts. The article Data Clustering: A Review [19] provides a wide overview of different techniques and ways to classify data into groups. They describe a variety of different clustering techniques, all with the goal to find patterns and assign data to a specific category that allows to work it. The clustering can be be distinct between hierarchical and partitional techniques. Different from a hierarchy, the partional methods doesn't produce a hierarchy but simple partitions. In their paper they describe different techniques for clustering different kinds of data such as image segmentation, object recognition, document retrieval and data mining. Different techniques can be useful for different kind of applications and therefore no perfect solution for everything is found. In another study, researchers from the University of Ouluu, Finland used the data of a wide range of different sensors to detect the context of the user [27]. They gathers data from a microphone, a three directional accelerometer, thermometer, light sensors and measured the humidity and skin conductivity. Using

naive bayesian classifiers to combine the gathered data and correlate them with samples. For example, they recorded the ambients of different environments, such as being in an elevator or the sound of a car or conversations. They got the features from their audio files by using algorithms from the MPEG-7 standardized metadata. Different environments have different key features in their ambients, such as constant noise (e.g. tap water) or peaks(e.g. conversation). Audio was their most valuable information, but for example the humidity was helping to detect very accurate whether the user is in- or outside. The results of the experiment show a very accurate detection of the correct context. Features like being inside or outside as well as detection whether rock or classical music was played were detected very well. Detection between walking and running or active and still were less accurate. The combined true positive rate was more than 90% and the true negative value was over 85%. Their usage of good test data placing their sensors in a good way helped a lot to get good results. In reality, when users have phones, they carry them in purses, pockets or their hand which makes it much harder to detect the current context. Context aware computing was already the topic of a great paper in 1995 written by a team from the Columbia University and the Xerox Corporation [39]. They describe a context aware system for an office environment. The systems uses the user's location, lightning, communication bandwidth and proximity to other users within the office in order to customize the application functionality depending on the context. This could for example include the displaying of experiment information, when the user is in his lab or showing the calendar when another person is in a close range. It can also be used to create context based reminders for specified locations, time, when seeing specific people or all combined.

2.5 Variable Quality Influences

The whole software quality is based on the cognitive performance of the software developer and the communication within the team. When the single developers write brilliant code, but don't know what the others do or need, the code can't work. On the other hand will the code quality still bad when the development team communicates perfectly but the individual programmers write bad code [31]. The following sections will provide an overview about the previous research of the individual factors in that area.

2.6 Team Communication

The differences between teams with high cooperating team-members against project teams with less communication have been investigated and discussed by Mary Beth Pinto and Jeffrey K. Pinto [33]. They tested the two different groups on performance in tasks and the psychological outcomes.

Several different factors have been tested and resulted significantly better in the high communicating group. They scored higher in resolving problems, brainstorming, progress review, obtaining information, gaining authorization to perform tasks and in receiving feedback. The low cooperating team did only get a better score in resolving conflicts, which is not surprising as fewer communication already avoid conflicts.

The importance of the communication in software teams is gaining more and more attention from companies within the last few years. The companies and teams came up with several ideas to improve the internal communication. Agile software engineering methods is the solution for a lot of teams and companies to reach the goal of a better exchange of information within the company. The concept is based on flexibility and responsibility within a software project.

Instead of having a the whole project scheduled and structured at the beginning, agile con-

cepts allow to react to problems and new information in a faster way /citechow2008survey. One of the most used and successful methods to work in agile teams is scrum.

In scrum, the tasks are separated in different phases that are called sprints. A sprint is a short time period in that a defined goal should be reached. This goal can be, for example a feature or a new component. After the sprint the team comes together again and decides about the next sprint and defines the next realistic realizable goals. In this way the team has a lot of responsibility about the project and a lot of freedom how they reach their goals within the sprint period. At the end of each sprint or a defined period, the team comes together for a retrospective to discuss the last sprint/s and how to improve the processes in the next period and if they change some methods such as the daily meeting. The daily meeting is done by some scrum teams, where every team member summarizes the achievements and problems from the previous day. This meeting can be useful or just wasting time. In order to find the best working and management patterns the teams can test different methods and discuss them in the retrospective. This dynamic changing and regular feedback is one of the reasons why scrum is used more and more in modern software teams [36] [31] even with downsides that the company needs to increase the trust in the employees and give up some control [35].

Another problem with the communication in teams comes with the increasing globalization and the internet. The ability that an employee can work from every part of the world with an internet connection brings the disadvantages that the software developers do not necessarily sit in the same room anymore or have their working place within a walking distance. Also allowing homeoffice for a few days a week is an option that employers provide their employees in order to be a more attractive and family friendly oriented. These changes also require new techniques to communicate within the teams. Communication can be done by using video conferences or email. However, both techniques have their disadvantages. A videocall needs to be scheduled and requires a good

internet communication and the problem with emails are the delayed response times and it is too easy for others to ignore an incoming email[9].

One possible solution is chat software which finding their way more and more in the daily communication in software development teams [20].

Slack is the most used tool for chatting at work. The great success in these new way to communicate shows in the ridiculous growth. The company Slack is the fastest growing Startup in the world. After just twenty month after its launch in February 2014, already more than 1.7 million people where using Slack [8].

2.7 Cognitive Performance

Looking at the individual programmer, the most important factor is obviously the cognitive performance of the individual person. This section shows the research in the circumstances and influences that can impact the performance in a long or short term.

2.7.1 Context Switching

Devin G. Pope and Ian Fillmore from the University of Chicago [34] inspected correlations in cognitive performance of students and the time between written exams. Depending on the schedule of the examinations, students from one year have a different amount of time between exams than students in different years. As a comparison they name the example of physical performance. If the body has a longer time to recover from one task, it performs the seconds task better compared to a shorter recovery time between these two tasks.

In this article they compare the scores of the students in their exams and the amount of days between the examination days. The study involves information about the students as class(Senior, Junior, Sophomore), Gender and their Race. They all were writing Advanced Placement (AP) Exams in the USA. Their results show that a longer break increases the

probability that the students pass the exams by 6-8%. The increasing of the performance is linear up to 10 days.

As one of the possible reasons for the outcome the researchers names fatigue which is caused by the exhausting task of studying and writing the exam. Another theory is that the last-minute preparations are important for good results but harder to realize when exams are closer together. Rogers and Monsell from the University of Cambridge [37] executed an experiment to find out how a context switch can influence the performance on cognitive tasks. It showed that a frequent context or task switching has a negative impact on the error rate and the reaction time of the participants for the tasks they did. Repeating this experiment for three days yielded that the practice has no positive influence on the error rate and thus shows as well that context changing is negatively influencing productivity and performance.

2.7.2 Working Environment

Improving the performance in Software Development can be done in a several different ways. One approach to improve the performance is the optimization of the working environment. Amabile, Teresa M., et al. [7] wrote about a conceptual model for increasing creativity in the work environment. Five key factors were described. The first two factors were, the encouragement for innovation and creativity as part of the company culture as well as according autonomy or freedom for the employees. Another key described the adequate availability of resources for a project which might affect people psychologically by the feeling to work on a valuable project. Also pressure at work was identified to increase creativity on a balanced level between excessive demands and boring routine. The last key factor in their model described the organizational impediments to creativity which could be caused by internal competitions. A study was designed to investigate two hypotheses: The influence of the model in high-creative projects vs low-creative projects is expected to

be much bigger. As well as obstacles scales are lower in high-creative projects compared to low-creative projects for workload with pressure and organizational impediments. Both hypotheses had clear result outcomes, which showed that beside the employee's itself, the management can significantly influence the level of creativity and innovation by forming the organization culture. The construction of the teams and definition of the individual roles can have a great impact on the creativity.

2.7.3 Arousal Effects

The cognitive performance can vary based on the context and the environment. When the body is in a relaxed state, the mind also slows down to save resources. It made sense back in the stone age because thinking was not as important as today. Cognitive performance was mainly needed in dangerous or unusual situations where the heart beat is faster to provide the brain and the muscles with more oxygen and a higher arousal than normally. Researchers from the Brunel University in the UK showed movie clips to participants in order to invoke different defined moods before the participants had to solve given debugging/coding tests. The results showed improvements in their score after the participants were confronted with high arousal video clips. Low arousal clips affected their performance in a negative way compared to neutral clips [26].

It is called the Yerkes-Dodson Law, which proclaims that a higher level of arousal leads to better cognitive performance. As caffeine also influences the arousal, it also can be used to boost the cognitive performance and is not just helping to wake up in the morning. Watters, Paul Andrew et. al. [44] found out that the average caffeine for the best cognitive results is an amount of 400 mg for one person(the amount contained in ca. 5 espresso shots).

When an arousal stimulation can be influencing the performance of a programmer, other factors that are effecting the mood could also have an impact in the quality of the written

software. Many people believe that, for example the weather has a strong influence in the daily mood of a person. Certainly, Denissen, Jaap JA, et al. [12] found out that the sunshine alone actually has no notable effect in the mood of the most of the people. Certainly, some individuals have a so called seasonal affective disorder (SAD). Their mood is indeed strong being affected by the seasons with fall and winter depressions.

However, they found significant correlations between sunlight, air pressure and precipitation on the tiredness of the participants. A reason for the influence of sunlight could be vitamin D3. The most of it is obtained through exposure to sunlight and it changes the level of serotonin which was found to be partly responsible for the mood of a human.

2.7.4 Diet

A very different, but probably the most important factor in the long term cognitive performance are temporary diets and the consumed food during the lifetime. The human brain needs good fuel to run properly. A wrong diet can strongly influence the incidences of cognitive problems as well as healthy food can positively influence healthy ageing [42]. Some eatables demonstrated positive effects on the mental performance when they were containing flavonoids like for example grapes, tea, cocoa and blueberries. Different to the previous influences, the diet and the lifestyle are less obvious in their consequences. Their impact is slowly showing over several years and it's hard to prove their effects and that they are the influencing factors.

Studies on several mammalian species have shown that food which is rich of flavonoids have beneficial effects on memory and learning, with the ability to support neurons and protecting them again stress-induced injury. These foods also decreases the chances of Alzheimer and dementia. Other studies have shown that flavonoid-rich groceries improves the blood circulation and correlates with the growing of new hippocampal cells. These cells are located in the brain region that is identified to be responsible for the memory.

Chapter 3

Design

In this chapter, I describe the design of the Android application for gathering the data as well as the sever side implementation to compute, store and provide information.

There will also be a short description of some additional tools. First, I will start with a brief description of the two components and will follow with my design decisions and my reasons for the choices.

3.1 Functionality Overview

The purpose of the software is to gather information from a mobile device of a participant while he/she is working on a programming task. Afterwards the application sends the collected data to a server for further processing and analysis. The participant also simultaneously submits the written code which code quality will be detected and then correlated with the processed mobile device information.

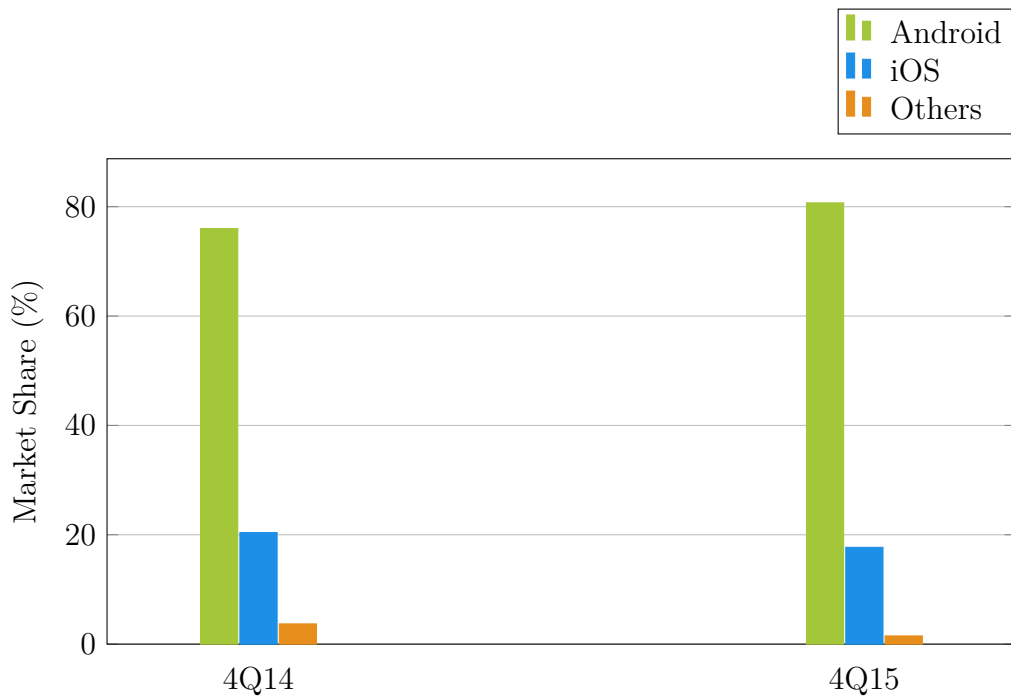


Figure 3.1: Smartphone OS Marketshare

3.2 Mobile Application Design

In quarter 4 of 2015 Android had a market share of 80.7% in smart-phone sales by operating system (see Figure 3.1). The trend also shows that the number increased from the last year [15]. Therefore I decided to realize the mobile application implementation for Android in order to be able to work with more users who have access to that application.

An alternative to the native implementation (e.g. iOS or Android) could have been a hybrid application. A hybrid apps is based on web-technology and using the advantage of resposive web design to be able to work with every aspect ratio and resolution on an mobile device. One way doing that would be by using a framework such as PhoneGap, wich internally creates a native webview applicationand just loads the hybrid JavaScript, HTML, CSS in it. Another software for cerating a hybrid solution is Titanium accelerator which itself is using native UI components. Both frameworks have

the advantage is the simple development and the OS independence. The problem with hybrid apps are the performance and limited accessibility to hardware components including some sensors [18].

The Android application make use of its build in sensors and information provided by the Android operating system. Different than iOS, Android is an OS that can be installed on different devices from different vendors and with different hardware components [16]. Thus the built in sensors which are clustered in motion sensors, environmental sensors and position sensors [17] can differ between the different devices. Components which are required for standard functionality such as making phone calls are more common than other sensors. For example the microphone for recording the users voice or the light sensor, which is used to detect whether the user has the phone at his ear can be found in almost every mobile android device.

3.2.1 App Architecture

The Android app is built based on the Model View Controller design principles. This design principle defines the interfaces between the three different parts, the Model, the View and the Controller and states the tasks and responsibilities of each part.

The Model is the data source and in this app represented by the SQLite Database and can only communicate with the Controller. The Controllers are called Activities in the Android Framework and is responsible for managing the Views, which are defined in XML files and then modified by the responsible Controller. To keep the code base clean and to avoid bugs, the communication is separated by the controller. The Model doesn't directly communicate with the View and can't update it. In the case of changes, the Model informs the Controller, which decides whether or not to update the view etc.

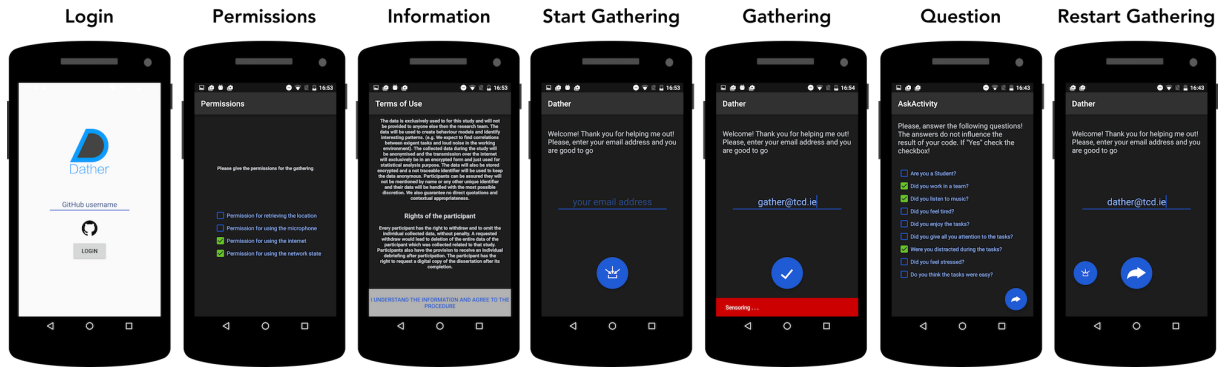


Figure 3.2: Android Views

3.2.2 User Interface

The Application contains of two main user Interfaces, the gathering view and the question view.

The gathering view is the control interface for starting and stopping the data gathering. Its interface changes depending the current state and the logical functionality and also provides an input field for the users email address.

After sending the gathered data to the server the application displays the question view. Here the user sees a number of questions and check-boxes to answer the binary yes and no questions. This interface also provides a send button to submit the answered questions to the Server.

In Figure 3.2 you can see the flow of the views in the applications.

3.2.3 Data Storage

For storing the gathered data entries, the app uses a SQLite database. SQLite uses SQL syntax which and is embedded in Android and well documented by Google. [43]. It is a very light weight database and provides an abstracted and easy way to store the values

in an object oriented environment. SQLite is storing the data unencrypted by default. In order to make sure the stored data is save and can't be read, the data is been encrypted as soon it's been gathered.

The database has a capacity of 1024 MB which is equal to 1,000,000 KiB and can store 2,380,952 entries of an average of 0.42 KiB (this value is calculated based on the used storage of 1,000 entries). Assuming that the maximum gathering get 30 entries per minute, the database can store 7,9365.07 minutes or 1,322.75 hours of the gathered data.

bigbreak Additional to the SQLite database, Android provides a way to store single entries such as the field with the email address of the user. The so called "shared preferences" are storing key-value-pairs persistently on the phone and can be accessed from everywhere in the application.

3.2.4 API

The gathered data and afterwards the answered questions will asynchronously send to the Server and written into the database. When the response will be received, the view shows a visual confirmation and moves on to the next step or in case of the questions back to the gathering interface.

The communication to the Server uses HTTP connection over a stateless REST-full service. The REST-Service is a centralized way to allow making entries to the database and use the computing power of the server. PHP is the programming language used server-side and is establishing the connection to the MYSQL database.

As this application just requires to send information to the server but not receive information, the data gets converted to JSON format and send to the server via a POST request. The server response with a success or fail and provides some additional information in case entries were added to the database.

3.2.5 User Information

For accessing the information from the mobile device, Android requires the user to granting the permissions for specific functionality such as using the internet connection of the phone or access information like the telephone book entries.

The permissions were given when the user by simply downloading the app from the PlayStore all in one place. However, since Android 6, the developer now is forced to ask for the permissions within the application itself [5].

Thus, the users with Android 6 or higher are provided by a additional interface which is specifically asking for permissions for Internet Access, access internet state information, using the device microphone and getting the location of the device.

Independent from the Android version the user needs to read and accept the terms of use at the first start of the application. The displayed text informs the user about the rights and what is happening with the data and information the user provides through this app usage.

Within the app the user is asked to enter his/her email address. The email address then is being hashed with the SHA256 algorithm to ensure the data will be anonymous and can't connected to the user.

Also the gathered data are stored encrypted in the SQLite database on the mobile device and just decrypted on a local computer after the researchers downloaded the encrypted entries from the SQL database on the server. That ensure that the files are not accessible in readable format at any time.

3.3 Server Design

The server is hosted by 1&1 Internet SE as a completely pre-configured server backend PHP version 5.6 and the MYSQL-server phpMyAdmin in version 4.1.14.8. The also pre-configured server file system can be accessed using SSH or via FTP.

3.3.1 Server Frontend Design

The information for the participants of the study can access information web pages and access the downloadable Android application.

The frontend is implemented in HTML5 and CSS3 and simply uploaded to the server. As everything is public available and accessible there was no need or using a framework or any further security implementation.

3.3.2 Server Backend Design

The backend is implemented in PHP with the MYSQL-server phpMyAdmin database. No external frameworks have been used to implement the basic REST-full service and the establishing of the database connection and the SQL queries.

3.4 Tools

The following tools are were used during the experiment primarily for the decryption of data to automate some processes.

3.4.1 Decrypting Tool

The encryption tool an executable program written in Java for locally encrypting the downloaded gathered data.

Its interface contains of an input field for the path of the downloaded data in JSON-format, a button for first reading and afterwards decrypting and a output text that shows the current state of the application. The tool writes the decrypted input values in a separate text file in the same directory of the input file with the same name but with the file extension .txt instead of .json.

3.4.2 Data Structuring Tools

I wrote a bunch of tools to work with the dataset. All the tools for this purpose are implemented in Python because it provides a very handy way to work with external files. The first tool allows to extract only the entries from a specific participant by providing the his/her Github username.

Another tool extracts the single values per entry with its timestamp and the last tool averages the results per minute and converts the data into a form that can directly be used in latex to draw plots.

Chapter 4

Implementation

In this chapter I will describe the implementation of the main application for Android, the server back-end and webpages as well as the tooling for working with the data.

4.1 Android

As mentioned before, the Android Application is for sensing data of the user and get environmental information. For This purpose I wrote an Android application which can gather these information.

Beside gathering the data using an App it is also possible to read the sensing information which are being recorded continuously as described in the approach of Zhu, Hengshu, et al. [47]. They are reading the device logs and get all the logged device more information about the apps being used etc. . Sandboxing is an Android security concept that only allows an app to access the data of the app itself and isolates the content for other applications. Thus it is impossible to access the device logs via an app without having physical access to the device.

In terms of the ideas for future usage of the app it doesn't make sense to require physical access to the device itself. Thus, the decision to use an App, installed on the users device,

is the best way to go for this purpose.

The implementation of the Android application has been done using the Android Studio IDE, which is provided free usage by Google, Inc. The code was written in Java, which is the official programming language for Android applications. Google also provides a variety of libraries and frameworks for user interface-Elements and basic functionality. For the user interface Android Studio has build in Solutions to either design the graphical user interface (GUI) using Java code or defining the elements in XML files.

4.1.1 User Interface

The user interface contains of two main views, the gathering view and the question view. There are two more views, one for asking the user for permissions and a second one for informing about the experiment and the terms of the usage for the application.

All view have a controller/activity java class which acts as the controller. The actual views contain of an activity XML and, depending on the complexity of the view, an additional content XML. Both are defining the UI-elements in XML tags and as well as their positioning within the view.

The colorscheme of the app is mainly a dark grey background with a combination of bright UI-Elements and simple lightgrey fonts for information texts.

Gather View

The gather view contains of an input filed for the users email address and a dynamic changing interface to for controlling the gathering and uploading process. The buttons are a blue circle shape with an icon for showing the functionality of the button itself. The Icons are a white shape without borders and designed to give a clear idea about the representing purpose of the button. Depending on the different states of the gathering process, the buttons change in functionality and look. In the first state, it only makes

sense to display the button that starts the gathering of the data. Once pressed a red bar with an information text on the bottom of the view indicates the running gathering process and the button that was starting the gathering changed to a new button for stopping the process.

A tap on the stop-button removes the red information bar disappears and the button changes its appearance and functionality to share/upload. At the same time, a smaller button appears on left hand side in the view which can restart the gathering process. After tapping on the share button, a green bar appears on the bottom and the question view opens.

Question View

The question view contains of a short information text that introduces the user to the new interface and a bunch of checkboxes for questions on it's left side. The questions can be either checked, to indicate a "yes" for the answer or can remain unchecked for "no". On the bottom of the view is another share button which sends the answered questions to the server once tapped.

The successful send is also being indicated by a green bar at the bottom and the question view is being replaced by the gather view.

Information View

This view contains a scrollview with a long formatted text. At the bottom of the scrollview is a button. With a tap on the button the user confirms the he/she read and understood the previous text and the user can go on to the next step which is either the permissions view or the gather view.

Permissions View

The Permissions view just contains of four checkboxes with it's descriptions, each for one permission. This view is just shown on devices with an Android version of at least 6.0. Once all the permissions are checked, a button appears which allows to go on. A tap brings the user to the gather view.

4.1.2 Data Gathering

The data gathering is managed by the gather class while the functionality is been managed by the sensor class. The most sensors can just be accessed by creating an instance of the single sensors. However, some, such as the environment volume have been customized individually in separate classes. The volume is no predefined sensor and needed to be created from the recording framework but without actually recording the sound. It is calculating the decibel from the current recording and just saved the gathered volume value. That ensures the privacy of the user and also doesn't need so much memory of the mobile device capacity.

As well as the volume measuring, the location has a custom implementation that uses the GPS or Wifi signal to calculate the current latitude and longitude of the device.

The app is gathering the data of each sensor every few seconds, between every 2 and 10 seconds, depending on the device speed. After receiving all the values from the sensors, microphone and Android OS, the app is generating a timestamp, adds the user ID to the entry.

This way to handle the gathered data make each singly entry independent from each other and can still be used in case of damaged data in some other entries.

4.1.3 Data Storage

Variables and temporary available resources are stored in memory during the runtime of the app. Anyhow, the memory can just store information as long as it's powered. The memory is also managed by the Android operating system and can be overwritten by other applications, once they are higher prioritized.

To store the entries and the user information permanently on the device on the hard disc its been stored in an SQLite database. The SQLite database handles the organization and keeps everything in a ordered form. It is also is resource optimized and allows easy access to the database from the applications.

The only data that is being stored permanently is the encrypted gathered sensor data. The permanent storage make sure that the data is not lost in the unlikely case of a crash of the application or a failing in sending the data to the server.

In order to save states such as the information weather the user already confirmed the he/she read and understood the terms of use, Android provides a method called Shared-Preferences. They can store single key-value pairs and are additional to the app-states used to store the users email address to avoid that he/she has to type it in every time the app restarts.

4.1.4 Security

In order to prevent that the participants can be identified by the user id because it is been generated by a SHA256 hash function that is infeasible to invert. In other words, the SHA256 algorithm generates a base16-String from the email-address of the user and there is no mathematical known way to recover the original email address in feasible time from the base16-String.

For encrypting the gathered data, the entries are independently getting encrypted before written to the database using a hybrid cryptographic procedure. Hybrid cryptography

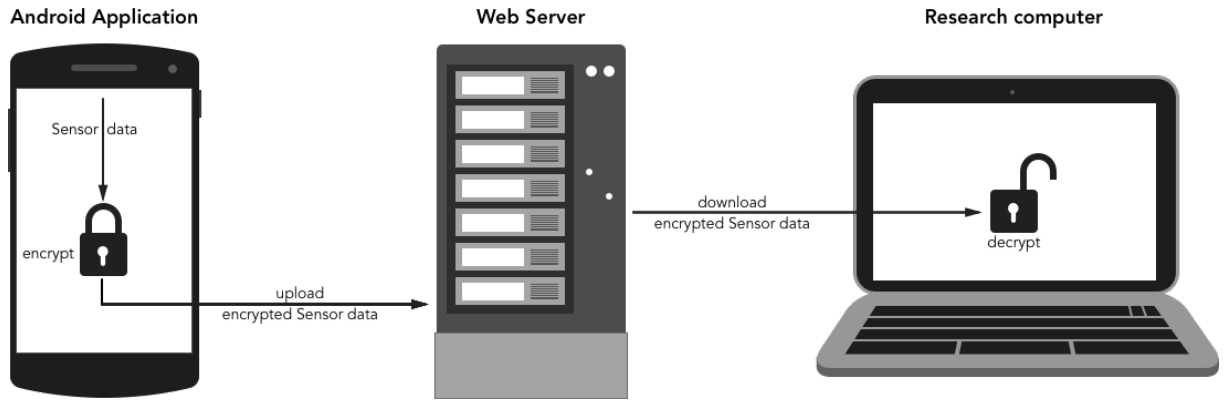


Figure 4.1: Security Dataflow

means the combination of using a the faster and performance friendlier symmetric cryptography (using the same key for encrypting and decrypting) and the slower but more secure asymmetric cryptography. In the asymmetric procedure also known as public-key cryptography, uses two different keys for encrypting and decrypting. A public key is used for the encryption of the data and the private counterpart is used to decrypt the data.

The encryption is the Android app works as follows:

A symmetric key will be generated every time the app starts using the AES CBC algorithm with an PKCS5Padding and a random SHA1 seed. This symmetric key will be used to encrypt the gathered data, while the symmetric key will added to each entry encrypted with the public key of a pre-generated RSA 1024 bit key-pair.

The private counterpart of the public key will later be used to decrypt the symmetric key. That symmetric key is then used to decrypt the entries. The decryption will happen with a separate written Java application locally on a computer.

Thus, a decryption within the applications is not possible because the functionality and keys are not even included.

4.2 Server

The server contains of three different parts, the back-end that handles the REST-full API calls, the MySQL Database and the web-pages for providing information to the participants of the study. In this chapter I will just write about the back-end and the web-pages because the Database design is already described in the previous design chapter.

4.2.1 Back-End

In the PHP script, the data from the POST gets extracted and decoded from JSON to an PHP-Array.

If the format of the data is correct, the script connects to the MySQL database and inserts the values into the corresponding table using SQL-Syntax. For each entry a counter is increasing it's value and after completing the insertion, the counter-value gets returned as a response argument. When something wrong happens, the script is responds with an error-code.

4.2.2 Webpages

The websites are implemented as simple as possible. They are completely static and only for displaying styled text and images. Therefore the implementation is only been done using HTML5 for the structure and CSS3 for styling the fonts, images and visual structuring.

Different fonts were embedded using Google-Webfonts from <https://fonts.google.com> [2] which are dynamically being loaded at the page load or from the browser cache.

4.3 Tools

The development of the following tools was necessary to work with the data that can be downloaded from the MYSQL database.

4.3.1 Encryption Tool

The encryption tool is a Java application that is written to decrypt the downloaded encrypted JSON-File of the gathered data. The simple tool is implemented in Java and is using the IntelliJ interface builder which is based on XML. First the tool read the input JSON-File and writes the beginning of the file into the output textfield.

Afterwards it is decrypting the symmetric AES-key using the asymmetric RSA-algorithm with the counterpart private-key to the public-key which was used for encrypting. Having the symmetric key allows to decrypt the whole input line by line using the AES decrypting algorithm. At the end the decrypted entries are written to a new created file and the filepath is been displayed in the text label.

4.3.2 User Separation Tool

This and the following three tools are written in Python. This tool reads the decrypted text file that has been created by the java decryption tool. First, it creates a SHA256 Hash from a Github username and compares the entries of the text file with it. It only takes the matching entries and writes them to a new text file.

4.3.3 Value Separation Tool

In this tool it can be defined which of the entries are needed to work with. For example the user can decide just to create an output file with the latitude of the participants location.

The selected data and its timestamp gets written in a new text file as well.

4.3.4 Latex Plot Syntax Creating Tool

This little tool is calculating an average for every Minute of the timestamp of the read text file. As the gathering saved a value every few seconds, it makes no sense to display all the values in a plotted graph.

The output contains the timestamp with the value for every minute in a syntax that can directly been interpreted by latex and the pgfplots library.

Chapter 5

Experiment

In this chapter, I describe the details about the experiment to find correlations between the data gathered from the mobile devices and the code quality. It shows the execution of the experiment as well as the usage of the gathered data and interpretation.

5.1 Crowd Experiment

In the experiment, participants are solving a programming task while the Dather Android App is running to record behavior and environmental factors. After the submitted code is been analyzed, it has been compared with the gathered data in order to find correlations between the participants performance and the information from the gathered data. You can see the flow of the experiment in Figure 3.1.

5.1.1 Setup and Execution

Every participant needs to have access to a mobile phone with Android version 4.4 or later. In order to take part at the experiment, a participant needs to install the Dather Application of his/her device. The Application can be downloaded from the website

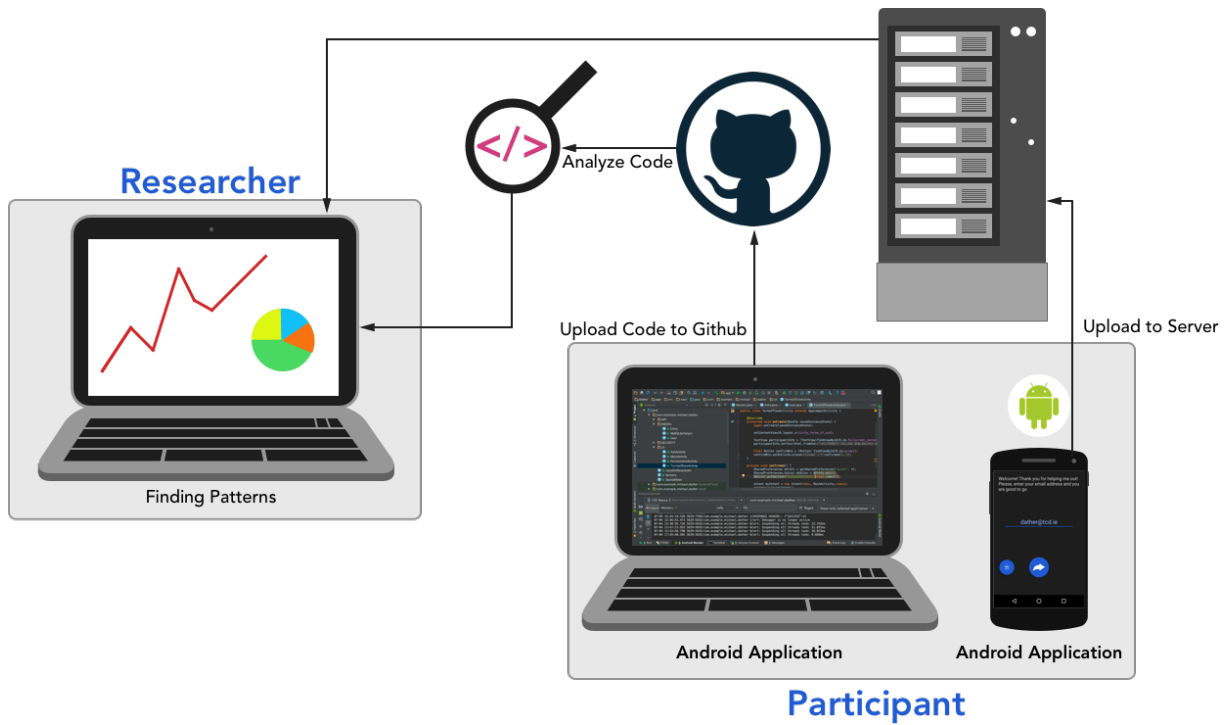


Figure 5.1: Experiment Execution

`http://frickm.de` when it's been accessed from the Android device.

After installing the downloaded apk-file, the participant gives permissions within the application to gather the data and allows the data to be used for research. As the last step before being able to start the experiment, the user needs to enter his/her email address. After setting up the application and accessing the website which contains the programming task, the experiment is ready to start. The participant runs the gathering process while working on the programming task.

After completing the task, the participant uploads the solution code to Github and sends a link of the Github repository from the Email address. The github account name can later be used to match the gathered data with the uploaded solution-code of the participant.

5.1.2 Classification

With the result values it might there might occur correlations between the entries and the coding quality. However, that approach is not using the full capability. In order to understand the values rather than just using them, it makes sense to interpret them and bring it into a context. Previous research results and also classifying controlled tested events using the gathered values will be described in further detail within the next paragraphs.

Indoor Outdoor differentiation

The brightness of indoor lightning is different from the brightness outdoors. Indoor environments are mostly receiving light from an artificial light source which flickers in a rate than can't be noticed by the human eye. Sadly the light sensor of the mobile devices is not precise enough to detect that flickering. Anyhow, also the luminance is different indoors and outdoors. Artificial lights are just not as powerful as the sun and it would require a ridiculous amount of artificial light sources and windows to create the same brightness within buildings as they are outside. As seen in the two tables 5.1 and 5.2 based on the lux from the light sensor it is possible to detect whether the device is indoor or outdoor by a high probability.

Common Light Levels Outdoor - Daytime	
Condition	Illumination in lux
Sunlight	107,527
Full Daylight	10,752.7
Overcast Day	1,075.3
Very Dark Day	107,527

Table 5.1: Common Outdoor Light Levels

Common and Recommended Light Levels Indoor	
Activity/Location	Illumination in lux
Warehouses, Homes, Theaters, Archives	150
Easy Office Work, Classes	250
Normal Office Work, PC Work, Study Library, Groceries, Show Rooms, Laboratories	500
Supermarkets, Mechanical Workshops, Office Landscapes	750
Normal Drawing Work, Detailed Mechanical Workshops, Operation Theatres	1,000
Detailed Drawing Work, Very Detailed Mechanical Works	1,500 - 2,000

Table 5.2: Common & Recommended Indoor Light Levels

Usage of Mobile Phone

The Y and Z axis of the 3D accelerometer can be used to detect whether the participant uses his phone. The simple classification picks up the change between the mobile device laying flat on the desk and the device being in a vertical position which is the position it would be when the user holds it in his/her hand. The graphic 5.2 shows shows the two states and the changes in the Y and Z-axis values.

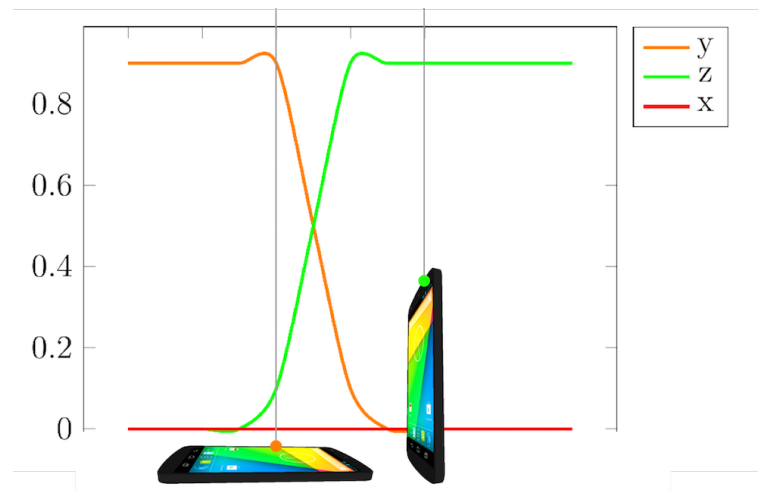


Figure 5.2: Android Views

Guessing the users location

In order to guess the location of the user, the location with the environmental noise as well as the detection whether the user is indoor or outside. The location accuracy depends on the way how it is been calculated which is either the network or GPS. However, it can vary and can't ensure a perfect detected location but using the noise and indoor/outdoor information can help to limit the results to less possibilities. When, for example the location shows a radius in an area with a library, a coffee shop and a public crowded square it's a high chance that the library is not an option in the case of a noisy environment. In order to detect whether the user is in the coffee-shop or the square, the light sensor can

detect whether the light value is in the outdoor or indoor brightness range.

Movement

The movement of the user can directly be seen by the steps he/she walks during the start of the gathering until the ending. The distance and the frequency shows if the user just walks to the fridge, toilet or somewhere close or actually walks from one place to another. Also the locations can indicate that. The location can also show whether the user was on public transport, on a train/car or an Airplane depending on the travel speed and from where the user started and where he/she arrives (airport, garage, train-station etc.).

Weather Conditions

With the location and the timestamps of the gathering and it is possible to get information about the local weather of the users location at the time when the gathering happend using the website <http://www.timeanddate.com/weather> [6].

Music

Using the environmental noise it is possible to find patterns that can be related to music. In general modern music has a very constant noise level rather than the dynamic classical music. The iTunes top 100 songs at July 14th 2016 have an average length of 3:39 minutes, the shortest song is 2:42 minutes and the longest 5:13 minutes long. In order to detect whether the participant is listening to music the volume should go down for 2-5 seconds between a track with a duration between 2:30 minutes and 5:30 minutes. A regular pattern with these attributes should indicate that the user is listening to background music while working on the coding task.

5.1.3 Questions

After the gathering process, the participant is asked to answer some questions:

- Are you a Student?
- Did you work in a team?
- Did you listen to music?
- Did you feel tired?
- Did you enjoy the tasks?
- Did you give all your attention to the tasks?
- Were you distracted during the tasks?
- Did you feel stressed
- Do you think the tasks were easy?

All the questions can either be checked to indicate 'yes' or leave unchecked for 'no'. The answers can help to clarify the classification or to get new additional contexts. Some of the questions are created based on the knowledge from previous work of researchers and their results that can possibly influence cognitive performance. In a long term, asking questions is not optimal. In the future the app is supposed to learn and slowly make the questions unnecessary. Currently there is a way to detect whether the user is listening to music by identifying patterns but the accuracy is not exactly known and therefore also asked as a question as well could it be that the user is wearing headphones. If the detection using the environmental noise is highly accurate, the question can be removed from the app.

5.2 Individual Experiment

The purpose of the second experiment is to find evidence of specific factors that influence the ability of cognitive thinking. Different isolated scenarios are been tested by a participant in order to find correlations between the specific environments. This experiment allows to test the factors in a more controllable environment but based on one individual person.

5.2.1 Setup and Execution

In this experiment a participant solved some cognitive tasks while being in a controlled environment in order to test the performance influences of isolated factors. Of course it is very unlikely or even impossible to test a factor in complete isolation one factor. There are always side factors that which are unavoidable. They could be for example the human itself, sudden unpredictable changes in the environment and of course the problem in keeping the factors of one part of the measurement equal to the factors of other measurements. To minimize these factors, the 'Dather' Android application helped to monitor the environment and remove recorded tasks where the environment information are too different of results which are were correlated with each other. However, with this problems in mind, the idea to measure changes in the cognitive performance of a person, was measuring the time of finishing a Sudoku game. The game, where the goal is to systematically add missing numbers in a 9x9 matrix, requires concentration and logical combining of numbers. The sudoku game was already used in previous research for measuring the cognitive performance [41] [46]. Another reason for using Sudokus is that they can be randomly generated with a specific calculated difficulty level to make sure that every Sudoku is equally hard to solve. A website [4] generated the Sudokus uses an engine which is part of the gnome-sudoku software [1]. A medium difficulty level and a limited calculated range of difficulty to +/- 0.02 of 0.5 was the base for generating the

Sudokus which were then printed on paper, one per page.

5.2.2 Scenarios

The following scenarios have been tested. Each scenario was performed 10 times to get a good mean which decreases the randomness in the experiment. In order to control the environment variables, a modified version of the Android app recorded the environmental light and volume and it was made sure that the values don't differ much to have a influence in the results. The experiments were executed over a several days in mixed up order to avoid that the training-process in solving the Sudokus can also influence the overall average of the outcomes.

Music

The scenarios to compare in this part the influence of two different types of music and as a control scenario no music at all. The participant did the Sudokus while listening to Spotify-Radio [3] 'Heavy Metal' and 'Classical' over headphones on a defined level of volume. In the control case without music, the participant was not wearing headphone but working in a very quite environment.

Coffee

In this scenario I wanted to test the influences of Coffee in the cognitive performance as discovered by Watters, Paul Andrew et. al. [44]. Simultaneously to their results I used a caffeine level of almost the value that they found out is the optimum for cognitive performance (400 mg). The whole experiment was executed in 5 days in a row with two tasks before, and two tasks after having a coffee. First, the participant solved the Sudokus without taking any caffeine for more than 16 hours, which is more than enough to make sure no other caffeine intake can influence to experiment [29]. Additionally the

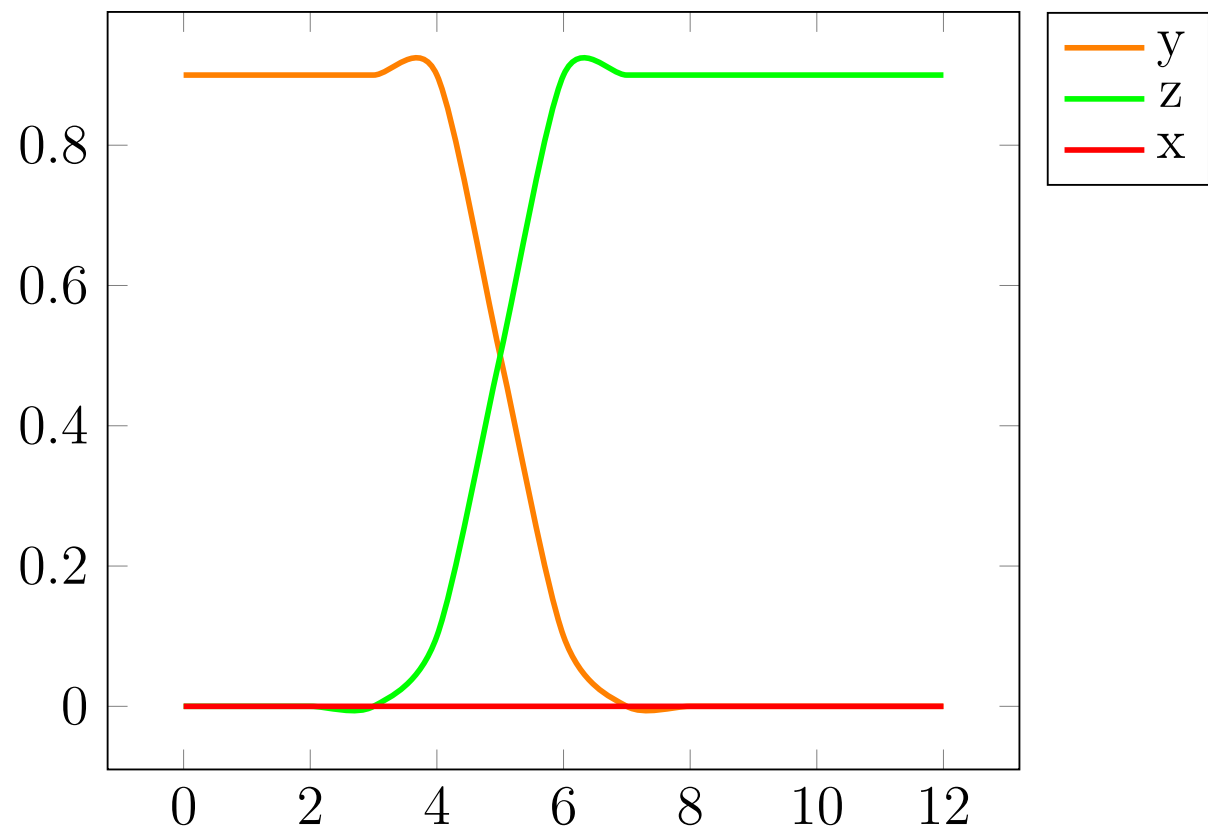
participant had the same breakfast every day before every experiment. For the second part of the experiment, the participant had the coffee drink that contained hot water with 5 espresso shots from Starbucks. The Coffee Franchise declares one espresso with 75mg caffeine each, which sums our drink up to 375mg at an amount of 5. After having the coffee, the participant waited 40 minutes for the caffeine to be absorbed [29] and started with the Sudoku.

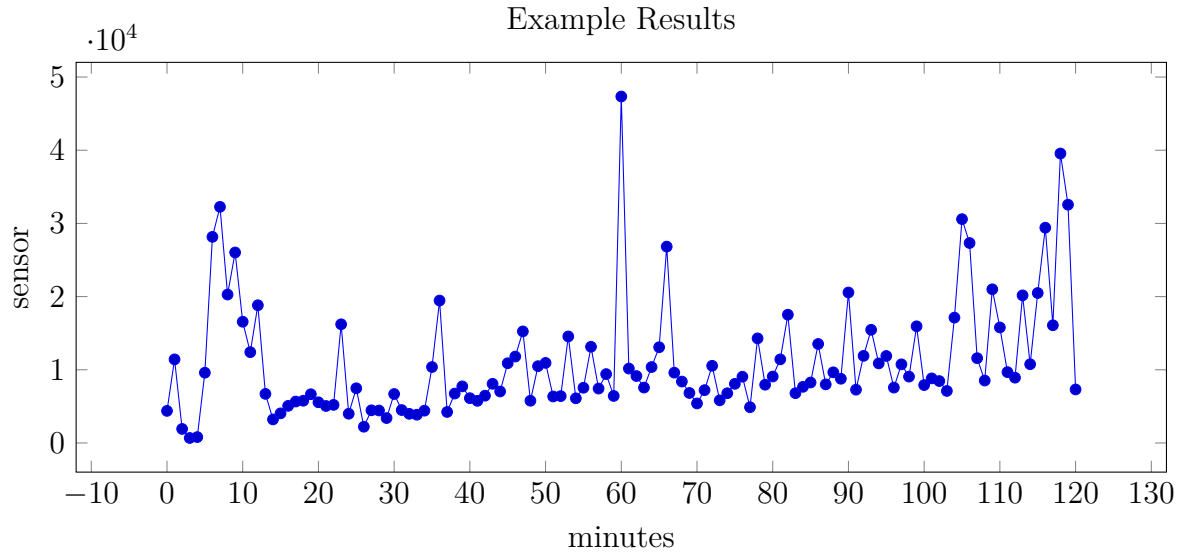
Chapter 6

Results

This chapter shows the results of the experiment and how the gathered data can be interpreted.

The results.....





6.1 Individual Experiment Results

Common Light Levels Outdoor - Daytime										
Condition	Duration									
Measurement	1	2	3	4	5	6	7	8	9	10
No Coffee	20:43	21:36	13:14	23:02	09:32	12:18	15:05	09:49	10:07	08:41
Coffee	12:39	22:32	09:00	26:34	14:23	16:21	25:18	16:16	14:47	18:15

Table 6.1: Cognitive Performance with Coffee

The table 6.1 shows the times of 10 Sudoku solvings of the participant. The results show that at the beginning, the participant was faster under the influence of caffeine and then later on the opposite. As the measurements were done on different days, the participant probably gained more practice in solving Sudokus, which shows the decreasing time to solve a Sudoku.

The average time, the participant needed without drinking coffee was **14:24 Minutes** and after consuming **17:37 Minutes** minutes per Sudoku. The participant needed 3:13 minutes or 22.34% longer after having a strong coffee.

These results are different than the findings in [29]. Their results showed that a similar

amount of caffeine increases the cognitive performance.

However, in our case the participant of the Experiment mentioned to feel fretful after the intake of the high amount of caffeine. That could be a reason for the lower performance of the subject. Thus, it is possible that a overdose had negative impact on the participant and lower amount of caffeine would have had resulted better.

Chapter 7

Conclusions

And a fancy conclusion...

7.1 Future Work

Appendix A

Abbreviations

Short Term	Expanded Term
AES	Advanced Encryption Standard
API	Application programming interface
APK	Android application package
CBC	Cipher Block Chaining
CSS	Cascading Style Sheets
GUI	Graphical User Interface
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
KLOC	Thousand lines of code
LOC	Lines of Code

Short Term	Expanded Term
PSP	Personal Software Process
PKCS	Public Key Cryptography Standards
RSA	Rivest-Shamir-Adleman
SAD	Seasonal Affective Disorder
SHA	Secure Hash Algorithm
SMS	Short Message Service
UI	User Interface
UX	User Experience
XML	Extensible Markup Language

Appendix B

Source Code

- Android Application for gathering the data

`https://github.com/MiChrFri/`

- Java Application for decrypting the data

`https://github.com/MiChrFri/`

- Python toolset for formatting the results

`https://github.com/MiChrFri/AnnaLize`

- Website and Backend

`https://github.com/MiChrFri/`

Bibliography

- [1] gnome-sudoku. <https://sourceforge.net/projects/gnome-sudoku/?source=navbar>. accessed on 20.07.2016.
- [2] googlewebfonts. <https://fonts.google.com>. accessed on 02.07.2016.
- [3] spotify. <https://www.spotify.com>. accessed on 23.07.2016.
- [4] sudoku-generator. <http://www.opensky.ca/~jdhildeb/software/sudokugen/>. accessed on 20.07.2016.
- [5] System permissions. <https://developer.android.com/guide/topics/security/permissions.html>. accessed on 02.07.2016.
- [6] Weather archive. <http://www.timeanddate.com/weather>. accessed on 14.07.2016.
- [7] Teresa M Amabile, Regina Conti, Heather Coon, Jeffrey Lazenby, and Michael Heron. Assessing the work environment for creativity. *Academy of management journal, publisher: Academy of Management*, 39(5):1154–1184, 1996.
- [8] Jeff Bercovici. Slack is our company of the year. here’s why everybody’s talking about it. <http://www.inc.com/magazine/201512/jeff-bercovici/slack-company-of-the-year-2015.html>, 2015. accessed on 15.07.2016.
- [9] Erran Carmel. *Global software teams: collaborating across borders and time zones*. Prentice Hall PTR, 1999.

- [10] Michael A Cusumano. How microsoft makes large teams work like small teams. *MIT Sloan Management Review*, 39(1):9, 1997.
- [11] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286, Lyon, France, 2012. ACM.
- [12] Jaap JA Denissen, Ligaya Butalid, Lars Penke, and Marcel AG Van Aken. The effects of weather on daily mood: A multilevel approach. *Emotion, publisher: American Psychological Association*, 8(5):662–667, 2008.
- [13] Norman E Fenton and Martin Neil. Software metrics: successes, failures and new directions. *Journal of Systems and Software, publisher: Elsevier*, 47(2):149–157, 1999.
- [14] Denzil Ferreira, Vassilis Kostakos, and Anind K Dey. Aware: mobile context instrumentation framework. *Frontiers in ICT*, 2:6, 2015.
- [15] Inc. Gartner. Gartner says worldwide smartphone sales grew 9.7 percent in fourth quarter of 2015. <http://www.gartner.com/newsroom/id/3215217>, February 2016. accessed on 08.06.2016.
- [16] Mark H Goadrich and Michael P Rogers. Smart smartphone development: ios versus android. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 607–612. ACM, 2011.
- [17] Inc. Google. Sensors overview. https://developer.android.com/guide/topics/sensors/sensors_overview.html. accessed on 08.06.2016.
- [18] Andreas Holzinger, Peter Treitler, and Wolfgang Slany. Making apps useable on multiple different mobile platforms: On interoperability for business application de-

- velopment on smartphones. In *Multidisciplinary research and practice for information systems*, pages 176–189. Springer, 2012.
- [19] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.
 - [20] Sirkka L Jarvenpaa and Dorothy E Leidner. Communication and trust in global virtual teams. *Journal of Computer-Mediated Communication*, 3(4):0–0, 1998.
 - [21] Yue Jiang, Bojan Cuki, Tim Menzies, and Nick Bartlow. Comparing design and code metrics for software quality prediction. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 11–18, Leipzig, Germany, 2008. ACM.
 - [22] Philip M Johnson. Leap: a ‘personal information environment’ for software engineers. In *Proceedings of the 1999 International Conference on Software Engineering*, pages 654–657, Los Angeles, California, USA, Mai 1999. IEEE.
 - [23] Philip M Johnson. Project hackystat: Accelerating adoption of empirically guided software development through non-disruptive, developer-centric, in-process data collection and analysis. *Department of Information and Computer Sciences, University of Hawaii, publisher: University of Hawaii*, 22, 2001.
 - [24] Philip M Johnson, Hongbing Kou, Joy Agustin, Christopher Chan, Carleton Moore, Jitender Miglani, Shenyan Zhen, and William EJ Doane. Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In *Proceedings of the 25th international Conference on Software Engineering*, pages 641–646, Honolulu, Hawaii, USA, 2003. IEEE Computer Society.
 - [25] Cem Kaner et al. Software engineering metrics: What do they measure and how do we know? In *In METRICS 2004. IEEE CS*, Florida Institute of Technology, Melbourne, Florida, USA, 2004. Citeseer.

- [26] Iftikhar Ahmed Khan, Robert M Hierons, and Willem Paul Brinkman. Mood independent programming. In *Proceedings of the 14th European conference on Cognitive ergonomics: invent! explore!*, pages 269–272, London, UK, Aug 2007. ACM.
- [27] Panu Korpipää, Miika Koskinen, Johannes Peltola, Satu-Marja Mäkelä, and Tapio Seppänen. Bayesian approach to sensor-based context awareness. *Personal and Ubiquitous Computing, publisher: Springer-Verlag*, 7(2):113–124, 2003.
- [28] John Laugesen and Yufei Yuan. What factors contributed to the success of apple’s iphone? In *Mobile Business and 2010 Ninth Global Mobility Roundtable (ICMB-GMR), 2010 Ninth International Conference on*, pages 91–99. IEEE, 2010.
- [29] Anthony Liguori, John R Hughes, and Jacob A Grass. Absorption and subjective effects of caffeine from coffee, cola and capsules. *Pharmacology Biochemistry and Behavior*, 58(3):721–726, 1997.
- [30] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, ISBN 0132350882, 9780132350884, Upper Saddle River, NJ, USA, 1 edition, 2008.
- [31] Nils Brede Moe, Torgeir Dingsøy, and Tore Dybå. A teamwork model for understanding an agile team: A case study of a scrum project. *Information and Software Technology*, 52(5):480–491, 2010.
- [32] Cindy Norris, Frank Barry, James B Fenwick Jr, Kathryn Reid, and Josh Rountree. Clockit: collecting quantitative data on how beginning software developers really work. *ACM SIGCSE Bulletin, publisher: ACM*, 40(3):37–41, 2008.
- [33] Mary Beth Pinto and Jeffrey K Pinto. Project team communication and cross-functional cooperation in new program development. *Journal of product innovation management*, 7(3):200–212, 1990.
- [34] Devin G Pope and Ian Fillmore. The impact of time between cognitive tasks on

- performance: Evidence from advanced placement exams. *Economics of Education Review*, publisher: Elsevier, 48:30–40, 2015.
- [35] Balasubramaniam Ramesh, Lan Cao, Kannan Mohan, and Peng Xu. Can distributed software development be agile? *Communications of the ACM*, 49(10):41–46, 2006.
 - [36] Linda Rising and Norman S Janoff. The scrum software development process for small teams. *IEEE software*, 17(4):26, 2000.
 - [37] Robert D Rogers and Stephen Monsell. Costs of a predicable switch between simple cognitive tasks. *Journal of experimental psychology: General*, publisher: American Psychological Association, 124(2):207–231, 1995.
 - [38] Philip E Ross. The exterminators [software bugs]. *Spectrum, IEEE*, 42(9):36–41, 2005.
 - [39] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90, Palo Alto, California, USA, 1994. IEEE.
 - [40] Thomas Schöps, Torsten Sattler, Christian Häne, and Marc Pollefeys. 3d modeling on the go: Interactive 3d reconstruction of large-scale scenes on mobile devices. In *3D Vision (3DV), 2015 International Conference on*, pages 291–299. IEEE, 2015.
 - [41] Rory Sobolewski, Richard B Reilly, Simon Finnigan, Paul Dockree, Kate O’Sullivan, and Ian H Robertson. Monitoring of cognitive processes in older persons. In *2009 4th International IEEE/EMBS Conference on Neural Engineering*, pages 132–135, Antalya, Turkey, 2009. IEEE.
 - [42] Jeremy PE Spencer. Food for thought: the role of dietary flavonoids in enhancing human memory, learning and neuro-cognitive performance. *Proceedings of the Nutrition Society*, publisher: Cambridge Univ Press, 67(02):238–252, 2008.

- [43] Lars Vogel. Android sqlite database and contentprovider-tutorial. *Java, Eclipse, Android and Web programming tutorials*, 8, 2010.
- [44] Paul Andrew Watters, Frances Martin, and Zoltan Schreter. Caffeine and cognitive performance: the nonlinear yerkes–dodson law. *Human Psychopharmacology: Clinical and Experimental*, publisher: Wiley Online Library, 12(3):249–257, 1997.
- [45] Niklaus Wirth. A brief history of software engineering. *IEEE Annals of the History of Computing*, 1(3):32–39, 2008.
- [46] Jie Xiang, Junjie Chen, Haiyan Zhou, Yulin Qin, Kuncheng Li, and Ning Zhong. Using svm to predict high-level cognition from fmri data: A case study of 4* 4 sudoku solving. In *International Conference on Brain Informatics*, pages 171–181, Beijing, China, 2009. Springer.
- [47] Hengshu Zhu, Enhong Chen, Hui Xiong, Kuifei Yu, Huanhuan Cao, and Jilei Tian. Mining mobile user preferences for personalized context-aware recommendation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4):58, 2015.