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

June 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

June 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

The abstract or summary or whatever...

Contents

List of Tables

List of Figures

Chapter 1

Introduction

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

Steve Jobs

Since the beginning of the computer, software was needed to be written. 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 [2]. Today, software is everywhere. The traffic is controlled by computers, security systems, nuclear power plants or just a messenger app on a mobile phone. 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 fire a missile without any command. [17] 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 be significant whether a company will be successful or just be one of many abortive start-ups with a good idea but a bad implementation.

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. People's performance can change by various different reasons. I will summarize the findings of previous studies that investigated different theories about influencing factors on cognitive performance.

In this dissertation, I will make use of the sensors and information provided by the user's mobile phone. I will write and Android application, that the user can install on his/her device, which 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. This sensoring will happen during the work an a provided programming task Afterwards the user gets asked to answer some questions such as information about disturbance or for example mood and more. In order to find patterns in behavior and environment which are influencing the quality of a programmer. I will cluster the gathered information into a specific behavior or context and compare them to find correlations with the code quality.

In the future, the app could become an every-day tool for developers and students. It

could be used hand in hand with project management tools, that requiere the exact times, a programmer worked on a project and it could simultaneously provide real time feedback about the code quality itself or destructive 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 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

Learning how to program is getting more and more important and still not a requiered subject in school. On the other hand, it is the computer with it's 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. Therefore it is very important that programmers produce high quality code. 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 themself and they might just used in for fun-project which were not created for public usage. However, what I want to say is that a programmer doesn't 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 replacement. When I started I had no practical experience in writing mobile applications. I had to maintenance the current code and add new features. As is 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 feedback of the code quality I could have learned a lot lot about

the coding itself and by today I would probably write much better code. 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 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 defining, measuring and improving code quality. I'm am also writing about previous research in cognitive performance and influencing working environments. Afterwards I will describe my design concepts of my approach followed by description of the implementation. The following chapters will inform about a study which is done using the developed Android application and the results. At the end I will evaluate the results and conclude the outcomes and findings.

Chapter 2 - State of the Art Chapter 3 - Design of Experiments Chapter 4 - Implementation Chapter 5 - Experiment Chapter 6 - Evaluation Chapter 7 - Conclusion

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 the developers. In my dissertation I aim to identify environmental influences that can improve or worsen the progress of gaining programming skills of Computer Science students. With the students as a key factor in future software engineering, I will analyze the behavior and performance influences in a environmental and psychological sense. [3] 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 feedback provided feedback helps them identify their weakness to improve their skills or change patterns. [10] [15] Also project managers have a great interest in details about the progress and the products quality to coordinate the schedules and resources. Early knowledge about potential problems can make the difference between the success or failure of a project. A good programmer nowadays is described as a person who can solve complicated problems by breaking them down in chunks that easy to understand and solve. The produced code is clean, easy to read and simple. [10] 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. The computing power grew rapidly and the increasing sizes of the most software projects with more and more people changed the requirements. From a research perspective it is very interesting to get an overview approaches from different years to gain a broader understanding. Thus the following 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 it's beginning, people wanted to measure and produce numbers to characterize code properties. The first metrics where 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 measuring the productivity, which can be used to for comparison with others. Software metrics is the term that is been used more then 30 years ago up to today. These days some of the most metrics are used to investigate the programmer productivity, the amount of bugs in relation to the amount of code, the initiall 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 reqieres. [13] 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. Also, 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 even for predictions of the software quality and it's reliability. [4] Several researchers investigated and developed approaches to improve the metrics and the results which they

are generating. Yue Jiang et. al. 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. [9]

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 about PSP and they developed different approaches to make students to adopt the PSP. Their first approach was originally described as âĂIJA Discipline for Software EngineeringâĂİ. 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. [12] 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. [11] The Appalachian State Uiversity 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 automatically, 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. [16]

2.3 Cognitive Performance

Improving the perfomance in Software Development can be done in a several different ways. A research group from the industry and the North Carolina State University investigated the influence of tooling and work patterns, used by the the developers [18]. They

created a tool which automatically recommended more efficinet development tools for programmers by analyzing the current work flows. The recommendation system monitored the patterns of the developer, compared the steps with the processes of other developers and made recommendations for adoption of more efficient tool. In order to understand the work patterns, the system needs to gather data from the computer of the developer. Different, existing tools like Mylyn Monitor, HackyStat and PROM were used for that purpose. Another approach to improve the development performance is the optimization of the working environment. Amabile, Teresa M., et al. [1] 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 indetified 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. A very different, but probably the most important factor in the cognitive performance is the consumed food. 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 for healthy ageing [19]. Some foods demonstrated positive effects on the mental performance containing flavonoids like for example grapes, tea, cocoa and

blueberries. Different to the previous influences, the diet and the lifestyle are less obvious. Their influence is shown over years and it's hard to prove it's effects. Studies on several mammalian species have shown that food that is rich of flavonoids have beneficial effects on memory and learning with the ability to support neurons and protecting them again stress-induced injury. It also decreases the chances of alzheimer and dementias. Other studies have shown that flavonoid-rich food improves the blood circulation and correlates with the growing of new hippocampal cells, which are in the brain region that is identified to be resposible for the memory. Even drinks or food that contains caffeine have an effect on the human brain. Based on the Yerkes-Dodson Law, a higher level of arousal leads to better cognitive performance. As caffeine influences the arousal, Watters, Paul Andrew et. al. [20] found out that the mean for the best cogintive results is an amount of 400 mg caffeine (the amount contained in ca. 5 espresso shots). Researchers from the Brunel University in the UK showed movie clips to invoke different defined moods of the participants before they solved given debugging/coding tests. The results showed improvements in the results after the participants were seeing high arousal video clips. Low arousal clips affected the performance negatively compared to neutral clips. [14] When the mood is influencing the performance of a programmer. Then factor effecting the mood 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. Denissen, Jaap JA, et al. [3] found out that the sunshine alone actually has no noteable effect in the mood of the most of the people but 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.

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 store and compute the information. 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.

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 [5]. 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

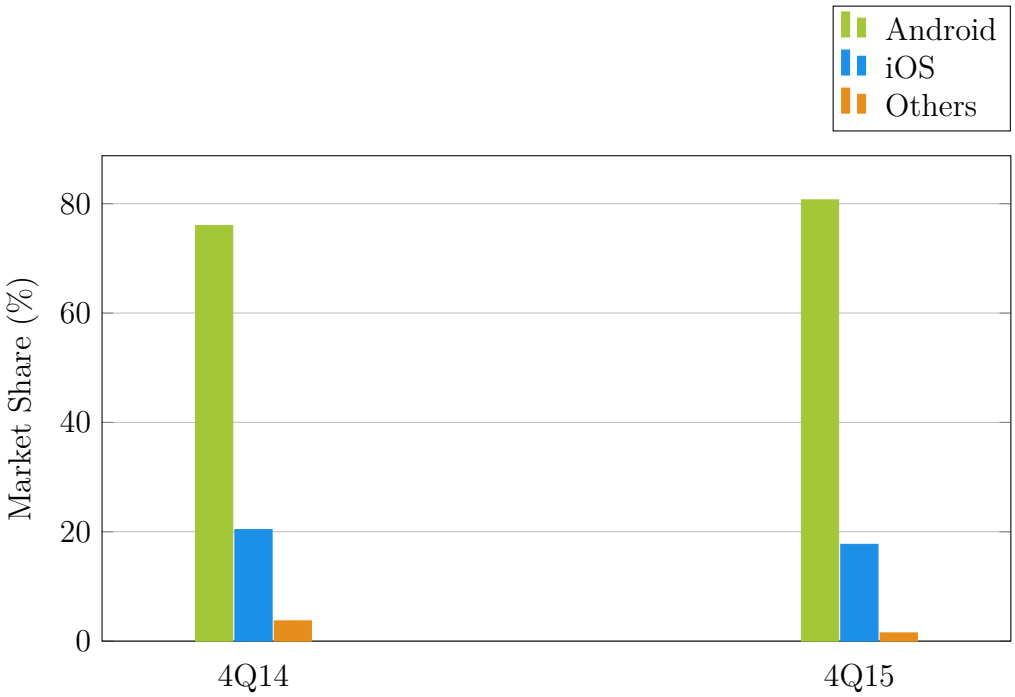


Figure 3.1: Smartphone OS marketshare

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 addvantage 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, HMTL, CSS in it. Another software for cerating a hybrid solution is Titanum 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 accessiblity to hardware components including some sensors [8].

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 of different devices from different vendors and with different hardware components [6]. Thus the buit in sensors which are clustered in motion sensors,

environmental sensors and position sensors [7] 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 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 Data Storage

For storing the gathered data entries, the app uses a SQLite datebase. SQLite uses SQL syntax which and is embedded in Android and well documented by Google. [**?**]. 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. 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.

3.3 Server Backend Design

Chapter 4

Implementation

4.1 Android

As mentioned before, the Android Application is for sensoring 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 sensoring information which are being recorded continuously as described in the approach of Zhu, Hengshu, et al. [21]. 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. The

4.1.1 Data Gathering

4.1.2 Data Storage

4.1.3 User Interface

4.1.4 Security

Hybrid cryptography using AES and RSA. RSA with the public key of a pre-generated 1024 bit key-pair.

The en and decryption uses AES with CBC and an PKCS5Padding.

AES generates a 128 bit key with a random SHA1 seed every time the app restarts.

Chapter 5

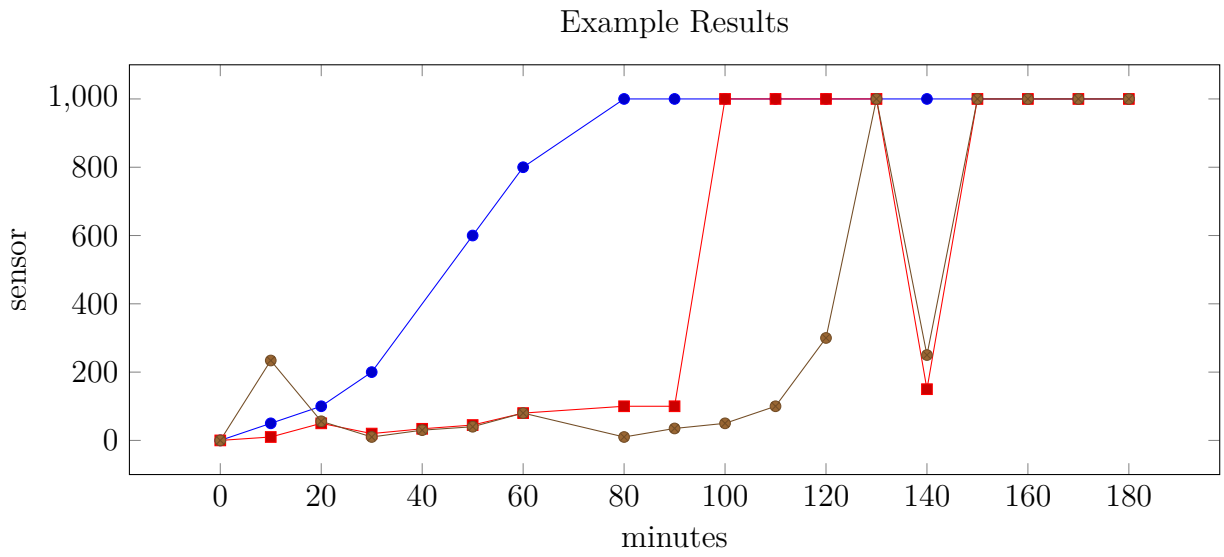
Simulations

The simulations.

Chapter 6

Results

The results.


Example Results
1,000
800
600
400
200
0
sensor
0
20
40
60
80
100
120
140
160
180
minutes

Chapter 7

Conclusions

And a fancy conclusion...

Appendix A

Abbreviations

Short Term	**Expanded Term**
LOC	Lines of Code
KLOC	Thousand lines of code
PSP	Personal Software Process
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
GUI	Graphical User Interface
UI	User Interface
UX	User Experience

Bibliography

[1] Teresa M Amabile, Regina Conti, Heather Coon, Jeffrey Lazenby, and Michael Herron. Assessing the work environment for creativity. *Academy of management journal, publisher: Academy of Management*, 39(5):1154–1184, 1996.

[2] Michael A Cusumano. How microsoft makes large teams work like small teams. *MIT Sloan Management Review*, 39(1):9, 1997.

[3] 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.

[4] 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.

[5] 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 06.08.2016.

[6] 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.

[7] Inc. Google. Sensors overview. `https://developer.android.com/guide/topics/sensors/sensors_overview.html`. accessed on 06.08.2016.

[8] Andreas Holzinger, Peter Treitler, and Wolfgang Slany. Making apps useable on multiple different mobile platforms: On interoperability for business application development on smartphones. In *Multidisciplinary research and practice for information systems*, pages 176–189. Springer, 2012.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Philip E Ross. The exterminators [software bugs]. *Spectrum, IEEE*, 42(9):36–41, 2005.

[18] Will Snipes, Vinay Augustine, Anil R Nair, and Emerson Murphy-Hill. Towards recognizing and rewarding efficient developer work patterns. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1277–1280, San Francisco, California, USA, 2013. IEEE Press.

[19] 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.

[20] 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.

[21] 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.