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

by

Michael Christian Frick, M.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 2017

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 1, 2016

Permission to Lend and/or Copy

Acknowledgments

Give many thanks and stuff...

Michael Christian Frick

University of Dublin, Trinity College

September 2017

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, 2017

Supervisor: Barrett Stephen

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]

1.1 Motivation

1.2 Aims

1.3 Road-map

Chapter 2 - State of the Art Chapter 3 - Design of Experiments Chapter 4 - Implementation Chapter 5 - Evaluation Chapter 6 - 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 analyse the behaviour and performance influences in a environmnetal 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. [6] [11] 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. [6] 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 importand 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 characterise 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 comparrision 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. [9] 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 softwares and scopes. Also maintenance and refactoring 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. [5]

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 " 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. [8] 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. [7] 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. [12]

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 [13]. 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 [14]. 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. [15] 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. [10] 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

More Content and Stuff

3.1 A section of more Stuff

Blah, blah, blah! Blah, blah, blah! Blah, blah, blah!

Blah, blah, blah! Blah, blah, blah! Blah, blah, blah!

- One
- Two
- Three

3.2 Another section of more Stuff

Blah, blah, blah! Blah, blah, blah! Blah, blah, blah!

Blah, blah, blah! Blah, blah, blah! Blah, blah, blah!

Chapter 4

Design

The design...

Chapter 5

Simulations

The simulations.

Chapter 6

Results

The results.

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

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] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.