AN EVALUATION OF THE USE OF SOFTWARE ENGINEERING PRACTICES BY COGNITIVE MODELLING RESEARCHERS

A MASTER THESIS

in

Computer Engineering

Atilim University

by

FURKAN KURTARAN

SEPTEMBER 2018

AN EVALUATION OF THE USE OF SOFTWARE ENGINEERING PRACTICES BY COGNITIVE MODELLING RESEARCHERS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

ATILIM UNIVERSITY

BY

FURKAN KURTARAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

SEPTEMBER 2018

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

_____

Prof. Dr. Ali Kara

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. İbrahim Akman

Head of Department

This is to certify that we have read the thesis "An Evaluation of the Use of Software Engineering Practices by Cognitive Modelling Researchers" submitted by "Furkan Kurtaran" and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Asst. Prof. Dr. Bilge Say

Supervisor

Examining Committee Members

Asst. Prof. Dr. Bilge Say _____

Asst. Prof. Dr. Atila Bostan _____

Asst. Prof. Dr. Güzin Tirkeş _____

Asst. Prof. Dr. Serhat Peker _____

Assoc. Prof. Dr. Erol Özçelik _____

Date: 24/09/2018

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Furkan Kurtaran

Signature:

ABSTRACT

AN EVALUATION OF THE USE OF SOFTWARE ENGINEERING PRACTICES BY COGNITIVE MODELLING RESEARCHERS

Kurtaran, Furkan

M.S., Computer Engineering Department

Supervisor: Asst. Prof. Dr. Bilge Say

September 2018, 59 pages

As an instance of scientific software, cognitive modelling is used to reveal how brains work in different levels of abstraction. Although there have been studies of software engineering practices in other domains of scientific modelling, cognitive modelling has not been inspected from a software engineering point of view. An international online survey with cognitive modelling researchers has been carried to pinpoint relevant points as well as to see whether there were any self-stated differences between developers and modellers; or between high level cognitive modellers and computational neuroscientists in their software engineering practices.

It has been found out that researchers in cognitive modelling, as in other scientific software domains, find frequent changes in teams to be problematic, specifying requirements to be hard, acknowledge the need for documentation and want to improve their software engineering practices. Participants find software engineering practices relevant, but their familiarity and level of use is lower, with the exception of version control and change management deemed both relevant and practiced. There are no significant differences between developers and modellers except for the observation that modellers stating themselves as more appreciative of validation. Similarly, no significant differences have been found between high

level cognitive modelling researchers and computational neuroscience researchers on their stated level of use of software engineering practices. However, researchers with larger team sizes use validation and verification more than those in smaller teams or working alone and larger user bases enhance the researchers' use of issue and bug tracking.

Keywords: cognitive modelling, scientific software, software engineering practices

ÖZ

YAZILIM MÜHENDİSLİĞİ YÖNTEMLERİ KULANNIMININ BİLİŞSEL MODELLEME ARAŞTIRMACILARI TARAFINDAN DEĞERLENDİRİLMESİ

Kurtaran, Furkan

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Dr. Öğr. Üyesi Bilge Say

Eylül 2018, 59 sayfa

Bilimsel yazılımın bir parçası olarak bilişsel modelleme insan beyninin çalışma şeklini farklı soyutlama seviyelerinde ortaya çıkartmaya uğraşır. Bilimsel modellemenin diğer alanlarında yazılım mühendisliğiyle ilgili çalışmalar yapılmış olunsa da, bilişsel modelleme yazılım mühendisliği bakış açısıyla değerlendirilmemiştir. İlgili noktaları belirlemenin yanı sıra; geliştiriciler ve modelleyicilerin, ya da yüksek düzeyli bilişsel modelleyiciler ile bilişimsel nörobilimcilerin yazılım mühendisliği pratikleri arasında bir fark olup olmadığını görebilmek için bilişsel modelleme araştırmacılarına uluslararası çevrimiçi bir anket düzenlenmiştir.

Bilişsel modelleme alanındaki araştırmacılar – diğer bilimsel yazılım alanlarında olduğu gibi – çalışma takımlarındaki sık değişikliğin problem oluşturduğunu, gereksinimleri belirtmenin zor olduğunu, belgelemenin gerekli olduğunu düşünmekte ve kendi yazılım mühendisliği pratiklerini geliştirmek istemektedirler. Katılımcılar yazılım mühendisliği pratiklerinin kendi işleriyle alakalı olduğunu düşünmelerine rağmen aşinalıkları ve kullanım düzeyleri versiyon kontrolü dışında düşük. Geliştiriciler ve modelleyiciler arasında modelleyicilerin doğrulamaya daha fazla değer verdiklerini belirtmeleri dışında önemli bir fark gözlenmemiştir. Benzer şekilde, yüksek düzeyli bilişsel modelleyiciler ile bilişimsel nörobilimciler arasında da yazılım mühendisliğini pratiklerinin kullanım düzeyi açısından bir fark gözlenmemiştir. Ancak, daha büyük takımlarda çalışan araştırmacılar doğrulama ve

sağlama tekniklerini küçük takımlarda veya tek başına çalışanlara göre daha fazla kullanmış, ve daha büyük kullanıcı tabanı olan uygulamalar araştırmacının sorun ve hata takibi tekniğini kullanımını arttırmıştır.

Anahtar Kelimeler: bilişsel modelleme, bilimsel yazılım, yazılım mühendisliği pratikleri

ACKNOWLEDGMENTS

I express sincere appreciation to my supervisor Asst. Prof. Dr. Bilge Say for her patience, motivation, and support. Her guidance helped me all the time and made this work possible.

I would also like to thank the rest of my thesis committee, and the participants of the survey for their contributions.

I am grateful to faculty and staff members of the Department of Computer Engineering at the Atilim University.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In order to support their research, scientists develop software. We take "scientific software" as software that enable computational modelling in place of or in addition to physical experimentation (Heaton, 2015). Such software that collect information, analyse large data sets, and create simulations is developed in various areas such as earthquake and geological engineering, nanotechnology, climate science, and computational chemistry. For example, in computational chemistry, the chemical reactions which are too dangerous, expensive or even impossible in real-world scenarios can be carried out using computational models. Generally, computational models allow scientists to study phenomena in different time scales or details than possible in physical reality. The exploratory nature of such models is different from traditional software which is mainly created to serve preset customer needs. It is known that quality of the traditional software is improved with the software engineering practices (Sommerville, 2011). Do the software engineering principles applicable in traditional software development benefit as-is scientific software development? Are scientists who develop code equipped with adequate software engineering background? Are they willing to learn and apply such principles? Heaton and Carver (2015) carried out a literature review that examines the claims about the use of software engineering principles in science. Ramakrishnan and Gunter (2017) suggested ten principles to create usable software for science. Based on their and others' work, as explained in detail in next section, some problems have been detected recurring from software engineering point of view in scientific software. Storer (2017) mentioned such problems as the "gap" between software engineering and scientific programming.

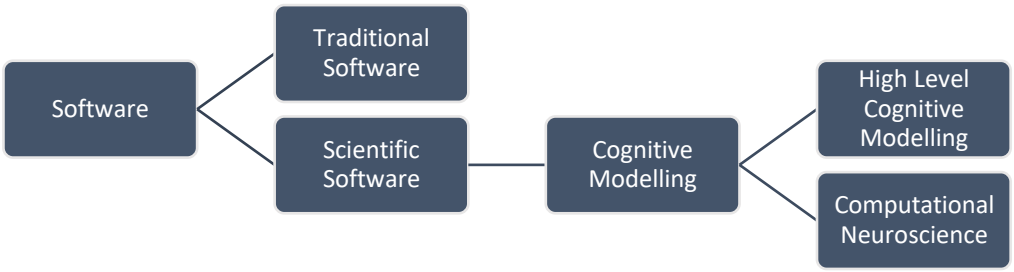


Figure 1. Taxonomy of Cognitive Modelling Software

Although there is extensive research on software engineering properties of scientific software, neither high level cognitive modelling nor computational neuroscience have been examined previously in particular. Computational cognitive modelling imitates the behaviour of the humans by developing corresponding models. Similar to high level computational cognitive modelling, computational neuroscience involves developing models – from a single neuron to whole brain network - to understand the brain functionality. These models are crucial to compare experimental results, to test scenarios - which may not be possible in real world - on humans or animals or to observe how cognitive or neuroscientific processes proceed. Figure 1 classifies the software terms that are going to be used in this study.

The goal of this thesis is to evaluate cognitive science and neuroscience modelling software development from software engineering point of view. For this purpose, our research questions are set as follows:

- RQ1: To what degree the cognitive and computational neuroscience modelling as scientific software development is similar or different from other scientific software domains in software engineering practices and problems?

Hypothesis 1: This is the main focus of the study. We expect the problems in cognitive/neuroscientific modelling to be compatible with other scientific areas and plan to investigate any diversions from a cognitive science perspective.

Due to the nature of cognitive modelling and computational neuroscience, modelling frameworks and cognitive architectures are platforms with theoretical assumptions on which models of various sizes can be developed. Thus, developers and modellers are sometimes the same people and sometimes not; but they both develop code, which brings us to our second research question:

- RQ2: What are the differences from software engineering perspective in terms of development practices between developers of the source code or modules of the

cognitive architectures, computational neuroscience frameworks or simulators versus model developers (developers of the models that work on those architectures, frameworks, or simulators)?

Hypothesis 2: Developers or developer-modellers are more familiar with and have experience in software engineering practices more than the modellers. The motivation for this hypothesis is that developers have to apply more complex design and coding practices by developing the core or extensions of the frameworks (Bican, 2007).

Commonly known as "end user developers" (Sanders, 2008), those who contribute to development of the scientific software and those who develop models written in code have not been distinguished systematically in the literature before. Therefore, we think this distinction will be useful in studying scientific software practices in other domains too.

Third research question is motivated by the observation that neuron or brain level modelling have different characteristics than high level cognitive modelling.

- RQ3: What are the differences from software engineering perspective between developers and modellers that work in high level cognitive modelling frameworks versus developers and modellers that work in computational neuroscience frameworks?

Hypothesis 3: Computational neuroscience framework developers and modellers comply with more software engineering practices than high level cognitive modelling framework developer and modellers. The basis for this hypothesis is that computational neuroscience frameworks are larger software with provisions for parallel threads and efficiency with larger user communities, more than the high level cognitive modelling software (Tikidji-Hamburyan, Narayana, Bozkus, & El-Ghazawi, 2017). This might have prompted such developers to be better equipped with software engineering techniques.

One aim of the researchers in the studies of software engineering practices of scientific software have been developing "domain specific methodologies" (Heaton, 2015) that might help overcome the problems detected in the literature. However researchers in different subdomains in the same domain might have different practices or level of competencies in software engineering practices. We will try to see if this is the case in cognitive science in different sub-domains namely high level cognitive modelling versus computational neuroscience.

An online survey is designed based on previous software engineering research evaluating scientific software and is carried out to answer above research questions. The online survey is a suitable instrument to research international community of developers and modellers

working with cognitive and neuroscientific modelling software in a quick and compact manner. They are reached through personal contacts, via forums, relevant conference participation lists, or mailing lists in cognitive modelling domain. Before conducting the survey, a pilot study is conducted to assess the questionnaire. Out of 178 candidate participants who entered the survey link only 85 were used in the data analysis. The surveys which did not fit a completeness criteria were discarded.

The rest of the thesis is as follows: Chapter 2 provides general information about cognitive modelling as scientific software; cognitive modelling, software engineering problems in scientific software, and common software frameworks and architectures for cognitive modelling. Chapter 3 presents the methodology used for the evaluation of the cognitive modelling software practices, explaining the design, pilot study, participants, and conduct of the survey. Chapter 4 presents the results of the survey and a discussion on these results. Chapter 5 is the conclusion of the study and states what can be done to move forward this study.

CHAPTER 2

COGNITIVE MODELLING AS SCIENTIFIC SOFTWARE

Scientific software is used to apply algorithms in scientific research. Gathering information, building large data sets, and simulating environments or systems are just some areas of application for this kind of software. To improve the quality of this kind of software, as in the traditional software, software engineering principles may be applied. The problems while applying software development life cycle principles in scientific software are discussed based on the previous studies.

2.1 Software Engineering Problems in Scientific Software

Most of the empirical work in studying the software engineering practices in scientific software have been employing surveys and interviews as methods. Some are written by scientists in that domain based on observations. Chronologically speaking, the interest in the interaction of software engineering and scientific software development have been flourishing in the last 10 years.

Sanders (2008) interviewed 16 scientific software developers and users to make recommendations regarding the software engineering practices. She stated that documentation is important but change of the requirements make it hard. Since the users can also be the developers, or can be in the same group, there should be a clear separation between developers and users within the domain. There is a tradeoff between usability and suitability of the software. Also, the usage of an iterative development process is preferable by all of the researchers.

Nguyen-Hoan et al. (2010) conducted an online survey with 60 scientific software developers. Participants were asked about their working environment such as their programming experience, development team sizes, produced documentation types, reasons to

test their code to reveal the situation in scientific software development. It is concluded that scientists do practice version control and documentation practices, but their testing practices should be improved. Similarly, Hannay et al. (2009) examined how scientists develop and use scientific software using an online survey. The results showed that scientists are mostly self-learners, prefer to use software which have larger user base, apply software engineering practices when work in large teams, and consider testing is valuable.

Carver et al. (2013) conducted a survey to collect researchers' perception of software engineering concepts and their level of use. They found that researchers do not have enough formal software engineering training and tend to be mostly self-learners. Although, these scientists believe their software engineering knowledge is sufficient for their work, their level of knowledge and use is low for the specific software engineering practices such as refactoring and code reviews. For all practices, researchers do not use practices which they are not familiar with and even though practices are found to be relevant, they are not used by the researchers. Even though agile methods are generally used in scientific software, Carver et al. (2013) interestingly found out that knowledge of agile method is low.

Wagner et al. (2015) discusses the challenges in simulation software engineering by sharing their own experiences. They emphasized that the motivation for developing software is to achieve new discoveries, the software itself has no value. Mostly, researchers do not have formal training in software engineering. Most of the team members are doctoral students and when they finish their education, they quit the project. Since there is a huge diversity of the software length of life, agile methods are appropriate for scientific projects. The correct results may not be known, testing is challenging. There is a trade-off efficiency/scalability and maintainability/portability in simulation software engineering.

Benureau & Rougier (2018) showed five characteristics of a scientific software. Through a piece of code, they illustrated what kind of problems might occur while developing software. It is pointed out that a code should be runnable whenever needed (re-runnable), produce same output over time (repeatable), allow obtaining same results by another researcher (reproducible), be used easily by other researchers (reusable), and allow modification to obtain promised results (replicable).

Johanson & Hasselbring (2018) carried out a literature survey to identify characteristics of scientific software development. These characteristics are gathered in three categories. Some characteristics arise because of the nature of scientific challenges such as unknown of the requirements and difficulty of verification and validation. Yet some other characteristics arise because of the limitations of the computers such as conflict in performance and maintainability.

Other characteristics arise because of the cultural environment such as lack of education of the researchers, underestimation of the value of the software and modern software engineering methods.

To sum up, unlike traditional software, the requirements are not known exactly in scientific software. Since the main purpose in science is to make novel discoveries, the needs to create a software is not considered precisely. In fact, scientist may find out the requirements while developing the software. If the desired outcomes are achieved, then they continue adding new features. Otherwise, they change or modify the idea until the goal is reached. In fact, this process fits perfectly with the agile development approach which is known as iterative development of the software. Due to the time pressure and documentation issues, researchers have been found to apply the agile methods knowingly or unknowingly.

Another issue in the development process is the value of the software. In traditional software development, the ultimate goal is to create a software. However, in science, software is considered as just a 'tool' used to achieve new discoveries. In addition to these, the team members within the scientific groups can be diverse. Most of the teams include master's and doctoral students. When these members finish their education, they may leave the project. This causes frequent changes within the group and may slow down the development process. Furthermore, team members may not have enough educational background in software engineering.

Although frequent changes within the members is common in scientific research, researchers do not pay enough attention on documenting their project. There are two reasons behind this: Firstly, most of the scientists are self-learners. Secondly, they think that they can ask their colleagues directly when they face a problem in software. Aside from these, it is not surprising that frequent changes in requirements make more difficult to document the development process. There are other situations that even if documentation is provided, the attached source code to the published research may not work or may not even be attached. Lastly, most of the time researchers may not know what the expected output should be which makes testing harder.

These key points that were extracted while searching literature were used in our questions in the survey as well as the key questions in the existing surveys as detailed in Chapter 3. Before introducing our survey, it will be useful to mention high level computational cognitive modelling software and computational neuroscience software as the domains to be studied.

2.2 Cognitive Modelling

A model is used to represent the certain elements in an abstract way. This representation enables us to figure out the functionality of phenomena in a simpler way. For example, surgeons may simulate a heart surgery using a model, which helps them to not only improve the process itself, but also avoid potentially dangerous mistakes and harm to the actual organ in case anything goes wrong. Completeness and faithfulness are two main properties of a model. Regarding the heart surgery example while simulating the operation, neither extra tools need to be added nor the structure of the heart be changed, offering identical circumstances and conditions to the actual settings. Cognitive models aim to understand how the mind works, how individuals make decisions, learn new things, and remember what they have learned by developing models that propose structures and processes to do so. Computational cognitive models help to answer such questions above mentioned by simulating and predicting the human behaviour with the help of algorithmic expressions to develop computer models to initiate detailed data processing. Computational cognitive modelling can be categorized as high level cognitive modelling and computational neuroscience.

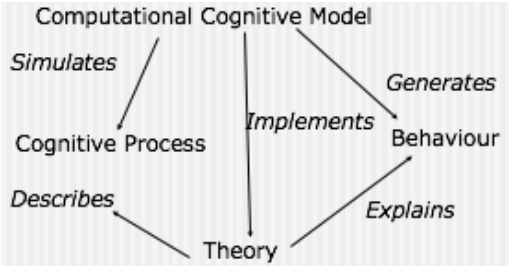


Figure 2. The Function of Computational Cognitive Models (Say, 2008)

Artificial intelligence, natural language programming, expert systems, and robotics are some areas that computational cognitive modelling is used. Figure 2 illustrates the function of the computational cognitive models.

2.2.1 High Level Cognitive Modelling

Although there are different approaches to categorize high level cognitive modelling, the one classification could be as follows: behavioural outcome modelling, qualitative modelling, and quantitative modelling. Behavioural outcome modelling simulates almost the

same actions as humans, qualitative modelling is concerned with representation and reasoning of qualitative human behaviour, and quantitative modelling generates the same quantitative behaviours as humans. Another classification is symbolic models, connectionist models, and symbolic-connectionist (or hybrid) models. Symbolic models claim that the mind is a symbolic system, consisting of symbols and symbol structures, and operates through manipulation. On the other hand, connectionist models (also called "parallel distributed processing") claim that the mind consists of neurons or units that operate simple processes, and that each of these neurons are connected to a larger network. Also, there are symbolic-connectionist models which are implemented through a combination of the two previous techniques as the name implies. These models implement symbolic structures in connectionist models. Any of these models may be used as either standalone or part of a cognitive architecture, which is build-up to simulate all the cognitive faculties and the organization of the mind in a high abstraction (Say, 2008). ACT-R, Soar, Emergent, Sigma, and LIDA are examples of the cognitive architectures which will be discussed later in detail.

2.2.1.1 Examples of High Level Cognitive Modelling Platforms

The present subsection summarizes some of the high-level cognitive modelling platforms that are updated regularly, publicly available, at least reasonably documented, and have mailing lists or forums. In this respect, ACT-R, SOAR, Emergent, Sigma, and LIDA are the five sample platforms which meet our criteria. In what follows, we briefly explain the basic idea of each system, application areas, developing process, and community of the platforms.

2.2.1.1.1 ACT-R

ACT-R is a hybrid cognitive architecture and stands for Adaptive Control of Thought-Rational. The main idea behind the ACT-R is that human brain consists of declarative and procedural knowledge. While declarative knowledge is represented as chunks, procedural knowledge is represented as productions. The ACT-R structure consists of modules and buffers. There are two types of memory modules in ACT-R: declarative memory that stores the facts, and procedural memory that stores knowledge about behaviour or procedures. Also, there are perceptual-motor modules that communicate with the environment. Moreover, buffers enable accessing to modules. Although there are parallel processes in the modules, buffers can only

execute one production at a time. For this reason, they also represent the current state of the system. The general structure of ACT-R can be seen in Figure 3.

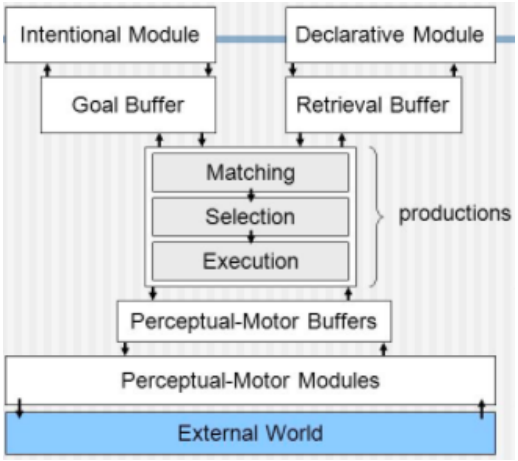


Figure 3. General Structure of ACT-R (Anderson et al., 2004)

ACT-R is written in Common Lisp and can be used for learning and memory, problem-solving and decision making, language and communication, perception and attention, and cognitive development. In addition, it is used for human-computer interaction, education, computer-generated forces, and neuropsychology[1].

2.2.1.2 SOAR

SOAR is one of the symbolic cognitive architectures and is short for states, operators, and reasoning. There are three cognitive levels of SOAR: memory level, that involves symbols of the knowledge; decision level, that is fundamental to decision operations; and the goal level, that is the combination of the previous levels. Furthermore, desired situation achieved in goal level.

[1] http://act-r.psy.cmu.edu/about/

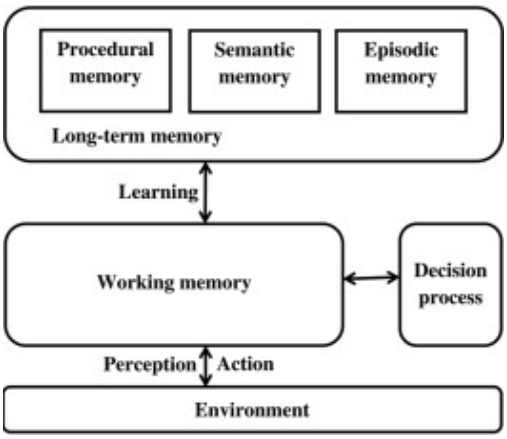


Figure 4. General Structure of SOAR (Trappenberg, 2010)

As seen in Figure 4, when the input comes from the environment, symbolic structures are added to the Soar's working memory. This memory is used as a global temporary memory and triggers knowledge retrieval from the long-term memory. Long-term memory stores the knowledge and has three independent parts: procedural memory, semantic memory, and episodic memory. Each piece of knowledge is stored as productions in the long-term memory and executions happen simultaneously when the conditions are satisfied in the decision process. Chunking and reinforcement learning are the two learning mechanisms of SOAR.

SOAR is written in a combination of C and C++, and used for puzzles and games, computer configuration, natural language understanding, simulated pilot training, steam, game agents, and scheduling[2].

2.2.1.1.3 Emergent

Emergent, previously known as PDP++, is a simulation environment for cognition that is written in C++ language. It has a 3D GUI that is rewritten using Trolltech Qt and Coin3d. Back-propagation, self-organizing, constraint satisfaction, and Leabra algorithm are the main functionalities of Emergent. Emergent provides visualization, easy usage of the models, and optimized runtime performance. Some areas in which Emergent is used are: robotic simulations (realistic human arm), sensory filtering for vision and audition, data processing, data analysis, and data generation[3].

[2] www.wiki-zero.com/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvU29hcl8oY29nbml0aXZlX2FyY2hpdGVjdHVyZSk

[3] www.grey.colorado.edu/emergent/index.php/PDP%2B%2B

2.2.1.1.4 Sigma

Sigma is a graphic-based cognitive architecture that claims to accomplish generic cognition, functional elegance, and sufficient efficiency. Sigma's cognitive system is composed of four levels that are lisp, graphical architecture, cognitive architecture, and knowledge and skills. Graphical architecture is the core part of the Sigma and it connects lisp and cognitive architecture together. Some application areas are: real-time needs of intelligent agents, robots, and virtual humans[4].

2.2.1.1.5 LIDA

LIDA (Learning Intelligent Distribution Agent) is a hybrid architecture that is based on cognitive cycles. The actions of the cycles can be categorized as perceiving, interpreting, and acting. These cycles form the higher-level cognition. Autonomous vehicles (LIDA-AV), arranging sailor's tour of duty for US Navy (IDA), and analyzing clinical data (LIDA Model – Max) are some application areas. The LIDA model is based on Global Workspace Theory[5] and framework implemented in Java[6].

As can be seen sample cognitive architectures for high level cognitive modeling focus on cognitive faculties and processes that are more related to higher mental faculties. They also have been used in applied artificial intelligence domains. Computational neuroscience frameworks, on the other hand, approach cognitive modeling bottom-up, dealing with the brain's physical levels.

2.2.2 Computational Neuroscience

Neuroscience is the scientific study of the nervous system. This study could be on a single neuron, which is a brain cell, or on the entire brain. As seen in the Figure 5, the nervous system is complex and involves multiple levels that require through understanding. In order to achieve this purpose, computational neuroscience researchers develop the necessary

[4] http://cogarch.ict.usc.edu/
[5] http://cogweb.ucla.edu/CogSci/GWorkspace.html
[6] http://ccrg.cs.memphis.edu/framework.html

computational models. It is an interdisciplinary science that collaborates with cognitive science, psychology, mathematics and computer science. These areas help to explain how the brain works. Developing models is a theoretical task and, hence, requires testing with experimental data to make it practically useful. In the meantime, this may also reveal some new features in brain functionality (Trappenberg, 2010).

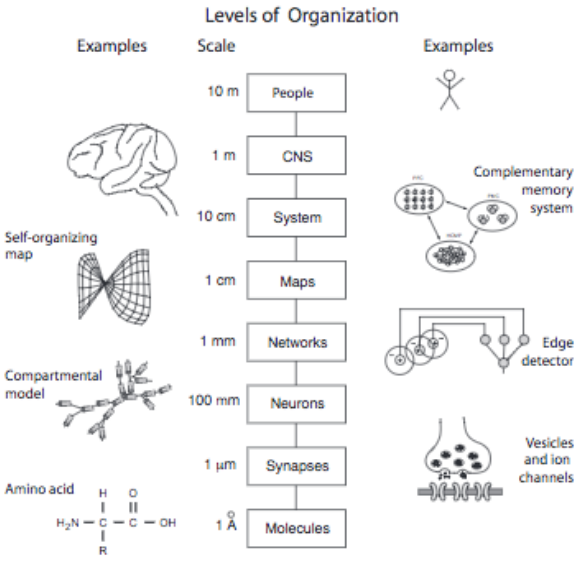


Figure 5. Level of Nervous System (Trappenberg, 2010)

Computational neuroscience is applied to fields such as single neuron modelling, sensory processing, development, axonal patterning and guidance, memory and synaptic plasticity, behaviours of networks, cognition and learning[7].

2.2.2.1 Examples of Computational Neuroscience Platforms

The present subsection summarizes some of the neuroscience platforms that are updated regularly, publicly open to use, and have forums or mailing lists. The five sample platforms meeting these criteria are OpenWorm, NENGO, Neuron, NEST, and Brian. The functioning principle, development processes, and area of use are explained briefly in the following list.

[7] www.scholarpedia.org/article/Encyclopedia:Computational_neuroscience

2.2.2.1.1 OpenWorm

OpenWorm is an open source project to create the first virtual organism in a computer. The aim is to simulate Caenorhabditis elegans (roundworm) which has approximately a thousand cells. It is the simplest nervous system and building it will lead to understanding more complex structures. Sibernetic (physics) and Gepetto (simulation) engines are created to model the worm. The source code is available on GitHub and the community arranges online meetings regularly. They have two mailing lists; one for general announcements and one to follow the daily activities[8].

2.2.2.1.2 NENGO

Nengo (Neural Engineering Object) is a graphical and scripting software package for simulating neural systems. Initially, it is written in Matlab, later in Java, and also Python as the last version. There are two underlying structures; Neural Engineering Framework (NEF) that defines connection weights between neurons, and Semantic Pointer Architecture (SPA) that represents symbols in Nengo. All of Nengo source codes and tutorials are available on its website, as well as the book "How to build a brain". They have a forum to communicate with the users and developers, and arrange summer school activities at the University of Waterloo[9].

2.2.2.1.3 Neuron

Neuron is one of the oldest, free, and open-source projects and is empirically-based on the simulations of neurons. Previously, it was written in HOC language, which is an interpreted programming language. Then, Python was adopted to make it more flexible. Installer, source code, documentation, and tutorials are available on the website. It is used in many laboratories and courses around the world. The most important property Neuron has is that it automatically creates familiar neuroscience concepts. This approach enables users not to deal with low-level concepts. Documentation and source code is available on its website, and a forum is set up to help the users and developers[10].

[8] www.openworm.org/getting_started.html
[9] www.nengo.ai/
[10] www.neuron.yale.edu/neuron/

2.2.2.1.4 NEST

NEST (Neuron Simulation Tool) is a simulator for spiking neural network models, and focuses on the dynamics, size, and structure of neural systems[11]. A Python-based user interface (PyNest) and built-in simulation (SLI) is available in the NEST network. Different sizes of neurons and synapses can exist, and these neurons can have multiple connections represented through adjacency lists. Some example models are: information processing, network activity dynamics, and learning and plasticity. The related source code and documentation can be found on its website and GitHub. The community uses a mailing list as a forum.

2.2.2.1.5 Brian

Brian is an open-source simulator for spiking neural networks, and is written in Python. The main purpose is simplicity, flexibility and to reducing users' development time. Unlike most of the neural network simulators, Brian focuses on a single neuron and enables users to create their neuron models by themselves. An interactive demo, installation, tutorials, example codes and email support list are available[12].

This brief overview gives a glimpse of the scientific software used in the cognitive modeling subdomains. The next chapter will detail the design and implementation of the online survey targeting the developers and modellers of these and other scientific software in cognitive modelling domain to better understand their software engineering practices.

[11] www.nest-simulator.org/
[12] www.briansimulator.org/

CHAPTER 3

METHODOLOGY

The design of the online survey and its procedural implementation is explained in the detail in this chapter, linking the survey to research questions. The whole survey can be seen in Appendix A.

Based on the research questions stated in Chapter 1, the following can be restated as objectives of the thesis:

- To understand the similarities and differences between software engineering practices in software developed for cognitive science and in other domains,
- To understand the differences between developers and modellers in cognitive science in terms of software engineering practices and backgrounds,
- To understand the differences in software engineering practices between researchers who work with high level computational cognitive modelling software versus computational neuroscience software.

3.1 Survey Design

In order to access the opinions and practices of the international community of high-level cognitive modelling and computational neuroscience, an online survey offers a plausible start with easier wide access and low cost in terms of time investment. To implement the questionnaire, Lime Survey [13] tool is used.

Previous surveys (Granados, 2000; Carver et al., 2013; Nguyen-Hoan et al., 2010) on scientific software are examined to shape the structure of the survey. Some of the questions/question

[13] www.limesurvey.org

groups are revised and adopted from these earlier studies with permission from authors[14]. Table 1 provides an overview of the questions of the survey. If a specific question is adopted/revised from a certain source this is stated along with the motivation for that question. Each part of the survey is explained in detail below the Table 1. In total, the survey includes 26 questions organized into five parts:

1. Introductory,
2. Demographics,
3. General Software Engineering Practices,
4. Scientific Software Development Opinions,
5. Feedback

Table 1. Overview of the Survey

Survey Sections & Question Numbers		**Adopted/Revised from**	**Motivation to be Asked in the Survey**
Demographics	Q1	-	To distinguish modellers and developers & To specify domains of participants
	Q2	-	To get the name of the modelling tool used/developed
	Q3	-	To learn educational background
	Q4	Carver et al., 2013	To learn how participants gained development skills
	Q5	Nguyen-Hoan et al., 2010	To learn years of experience to develop software
	Q6	Granados, 2000	To get participants' job description
	Q7	Nguyen-Hoan et al., 2010	To get participants' team size
	Q8	Nguyen-Hoan et al., 2010	To get participants' user base size
General SE* Practices	Q9 – Q16	Carver et al., 2013	To check if community is familiar with software engineering practices
SS Development Opinions**	Q17 – Q25	-	To allow community to indicate opinions about software development progress

* Software Engineering, ** Scientific Software

[14] One survey question was adopted from the survey by Granados, who was not accessible for permission.

The first part is where the questionnaire is introduced to participants. The purpose and the coverage of the survey is explained along with expected duration and information on how to complete the survey. The participants are informed that they can voluntarily provide their contact information, if they would like to participate in a book check gift draw which is provided as a motivator to complete the survey. The anonymity of survey responses is confirmed. This part acts as the informed consent form and the participants are accepted as giving consent by starting the survey.

The second part includes demographics related questions where general information about the participants are collected. Participants' development experience, educational background and training, years of software development experience, job description, team size, and user base size are asked to specify their profile. There are eight questions in this section. In Q1, the participants are asked to describe their software development experience in cognitive modelling. The participants can indicate whether they work in high level cognitive modelling or computational neuroscience as well as whether they are a developer or modeller. The developers are further refined into the developer of the core parts of an architecture or a framework or a developer of extensions/models of an architecture or a framework. The participants can choose all that apply. This question not only helps to distinguish modellers and developers but also their working area in the domain. This question will enable the participants to be characterized into groups for the hypothesis stated in the introduction: Developers are more likely to be familiar with software engineering principles than modellers and participants in computational neuroscience area are more likely to be familiar with software engineering principles than participants in high level cognitive modelling area. In Q2, name of the main cognitive modelling framework/architecture developed/used is asked to cross check with the answers given to Q1. Q3 is a yes/no question that asks if any formal degree that participants have major computer science or software engineering content. In compliance with our first research question, our sub-hypothesis is the participants with formal degree with major computing content background are more likely to apply software engineering principles than other participants. This survey question is presented to be able to check this hypothesis which has not been studied in literature before, although, formal training in computing has been found to help applying SE principles (Storer, 2017). In Q4, the participants are asked to rank how they obtain their SE development skills from different sources (Courses in Formal Education, Attending Training Course, Co-workers, and Learned on My Own). The answers of this question are again used to cross check whether training or formal courses affect general software engineering practices. In Q5, active years of experience in developing software is

asked. As part of the hypothesis that cognitive science domain is similar to other scientific domains it is expected that participants who have more experience developing software apply software engineering principles more than participants who have less experience (Nguyen-Hoan et al., 2010). In Q6, participants' job description is asked to examine whether they perceive themselves more as a researcher or a developer or whether they exclusively work in software development or maintenance. In Q7, software development team size is asked. Hannay et al. (2009) hypothesized that an increased team size affects positively for need for good software engineering practices, but they only found an effect in software requirements and software maintenance. We use this question to cross check as to whether there are significant differences in answers given to questions in the third part with respect to team size. The expectation is that the participants who work in larger groups apply software engineering principles more than the participants who work in smaller groups. In Q8, user base size of the developed software is asked. Larger user bases have been found to bring challenges in testing and debugging in Hannay et al. (2009). Thus, we want to see whether the larger user base, the more software engineering principles are applied.

The third part is related to general software engineering practices: general knowledge of software engineering life-cycles, documentation, requirements management, verification, validation, version control/change management, issue/bug tracking, and use of agile method. Each term has a brief explanation taken from Sommerville (2011) to prevent misunderstanding. The participants are asked to rate from "none" to "very-high" (None, Low, Medium, High, Very High) how they see each practice in terms of relevance to their work, personal familiarity, personal level of use, team familiarity, and team level of use. Those participants who do not develop software as a team could leave team related lines empty. This matrix-like organizations of questions are adopted from Carver et al. (2013). Carver et al. (2013) had more specific questions about design and testing techniques; whereas we concentrate on only the common and general issues also underlined in other sources (Sanders, 2008; Storer, 2017). This part of the questionnaire forms the basis of our statistical analysis for all our hypothesis with respect to categorizations we obtained in the demographic section as explained above.

The fourth part, named as Scientific Software Development Opinions, participants' opinions are asked for various finding in scientific software development. Five different options (I strongly agree, I agree, Neutral, I disagree, I strongly disagree, No idea) are presented for each item. The findings are extracted from the literature on cognitive modelling and scientific software development (Benureau & Rouiger, 2018; Bican, 2007; Heaton, 2015; Storer, 2017; Sanders, 2008; Taatgen et al., 2015). Some of these opinions relate with the practices questioned

in part 3 of the survey and have been found to be problematic in cognitive modelling or other domains of scientific software development. In addition, performance, usability and team member changes which are again observed to be problematic in the literature have been probed in separate questions. Two other concerns namely whether it is problematic to separate the theoretical model from the actual code and whether code reviews should be carried out in peer reviews in publishing form separate questions again inspired from the literature. This section ends with a question that measures whether participants believe learning more about software engineering practices would improve their software development work.

The last part of the survey allows participants to leave a comment on any of the questions or software engineering practices in general. The participants are thanked for their time and asked if they would like to receive more information about results or participate in further studies. Moreover, participants are informed about gift check and asked if they would like to participate in the draw. Such participants are given to the option to enter their contact details.

3.2 Pilot Study

Before conducting the survey, a pilot version of it was sent to the three experts through personal contacts. One of them is experienced in survey-based research, another one has PhD in cognitive modelling, and last one has PhD in cognitive science whose specialization in computational neuroscience. They provided feedback on the readability, relevance, and timing of the questionnaire. The Institutional Ethics Committee Approval, which can be seen in Appendix C, was obtained in this period and survey remained online for a week for the pilot study. In light of the feedback from pilot study participants, the questionnaire was revised as follows:

- Some changes on the interface made and more explanations added to increase usability,
- More explanation is added to clarify some parts of the questions,
- Some of the questions and answer options are updated.

3.3 Participants

After the revision of the questionnaire, an invitation e-mail was sent to the prospective participants from international community of cognitive modelling and computational neuroscience. The content of the invitation e-mail can be seen in Appendix B. Roughly

speaking, 1300 individual e-mail were delivered successfully to those who work in cognitive modelling area. These e-mails were gathered through authorship and snowballing citing relationships obtained from the recent conferences in these domains. In addition to these individual e-mails, the survey invitation e-mail was shared with following user groups/forums/e-mail lists:

- In High Level Cognitive Modelling:
    - ACT-R users mailing list,
    - Soar users mailing list,
    - Emergent users mailing list,
    - LIDA community,
    - Open-NARS google forum,
    - CLARION community,
    - 4CAPS community,
    - EPIC community,
    - Related Facebook groups (Cognitive Architecture, Cognitive Modelling Group, Cognitive Science Society etc.)

- In Computational Neuroscience:
    - OpenWorm community,
    - Nengo forum,
    - NEST users mailing list,
    - NEURON forum,
    - Brian google group,
    - Related Facebook groups (Theoretical and Computational Neuroscience, Cognitive Neuroscience Society, Neuroscience Researchers etc.)

In total, 178 people clicked the survey's link; 93 of them did not complete the whole survey, so 85 usable responses were collected. Apologies were stated for unintended duplicate invitations. As the coverage of the community was adequate and due to the time constraints of thesis, reminder e-mails were not sent. The survey was available for about a month, between July, 2018 and August, 2018.

3.4 Survey Conduct

The participants are welcomed with an opening page which introduces the survey as described in Chapter 3.3. After reading the presented information, the participants are prompted to click the "next" button, eight demographic questions as explained in section 3.3 are shown to the participants all in one page. When this part is done, they are prompted to click the "next" button again and general software engineering practices section is shown, all eight questions in one page. After the completion of software engineering practices section, participants click the "next" button and scientific software development opinion questions are shown, all in one page. After the completion of the opinion section, they click the "next" button for the final section of the questionnaire in which they are encouraged to provide further opinions as comments in text box. In all stages of the survey, a progress bar enables the participants to see their progress. At the end, they are asked to state if they would like to receive more information on the results, or participate in further studies, or participate in Amazon book check prize draw. The participants who click "yes" for these optional parts are asked to give their contact information. After clicking the "submit" button, submission is done. Finally, last page of the survey is shown. The participants are thanked for being a part of this study. References of the previous surveys from which some of the questions were adopted and contact information of the researchers can be seen on this page.

All responses are treated anonymously, none of the questions are forced to be mandatory to ease the conduct of the survey. On average, it takes about 10 minutes to complete the whole survey.

In the next chapter, the results of the survey and a discussion on these results are presented.

CHAPTER 4

RESULTS AND DISCUSSION

This survey was conducted with international participants in cognitive modelling and computational neuroscience community to study the following main research points:

- To compare and contrast the software engineering practices in other scientific domains versus cognitive science,
- To compare and contrast the software engineering practices of developers of scientific software versus modellers that use those software within the cognitive science domain,
- To observe any differences in software engineering practices within the two sub-domains of cognitive modelling: high level cognitive modelling versus computational neuroscience.

To this end, the results will be presented in two sections using descriptive and inferential statistics. The results will be followed by general discussion and limitations of the current study.

The results were analyzed with R programming language for statistical computing and graphics[15]. The results were analysed with data from 85 participants. All participants completed all the questions except for the team related questions in general software engineering practices section for those participants who did not work in a team and two participants in Q4 as explained in 4.1.4.

[15] https://www.r-project.org/

4.1 Descriptive Statistics

The descriptive results are presented in this section for all the questions in the survey. The percentages given refer to 85 participants unless stated otherwise.

4.1.1 Software Development Experience

In this question (Q1), participants were asked to state their role and domain. There are 41 (48.2%) participants in high level cognitive modelling area, 11 (26.8%) of them stated their role as modeller and 30 (73.2%) of them stated their role as developer, and 44 (51.8%) participants in computational neuroscience area, 5 (11.4%) of them stated their role as modeller and 39 (88.6%) of them stated their role as developer. Out of 30 developers in high level cognitive modelling 19 of them stated that they developed main architecture or framework, while 11 of them stated extensions or optional models. Out of 39 developers in computational neuroscience 32 of them stated that they developed main architecture or framework, while 7 of them stated extensions or optional models. The two categories were treated as one category to ease further statistical analyses. Note that, the participants that indicated their roles as both modeller and developer, 59 participants, were categorized as a developer, since they had direct software development experience in the framework or architectures. The distribution of the participants can be seen in Figure 6.

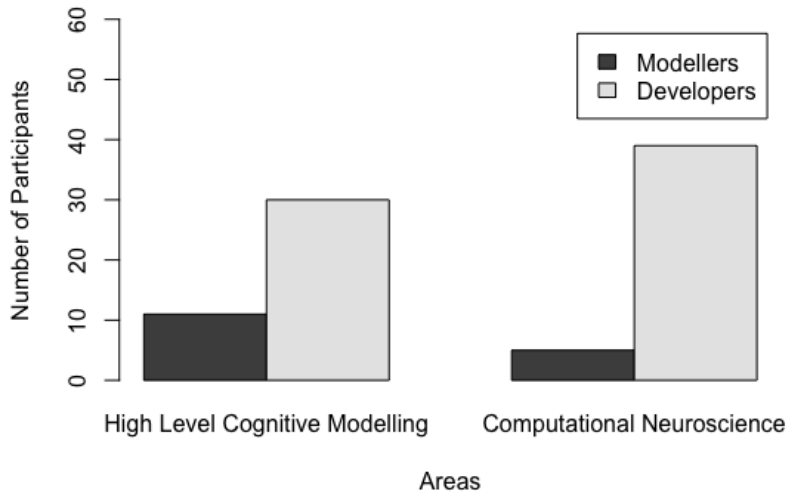


Figure 6. Software Development Experience

4.1.2 Name of the Tool Developed or Used

In this question (Q2), participants were asked to state the name of the main cognitive architecture/computational neuroscience framework that they developed or used. In high level cognitive area ACT-R (12.9%) and Soar (12.9%), in computational neuroscience area NEST (9.4%) and NENGO (7.1%) are the two most developed or used ones. The frameworks and architectures stated are compatible with the sub-domain choices made in Q1.

4.1.3 Having Formal Degree in Computing

In this question (Q3), participants were asked to state whether they have formal degrees with dominant computing content or not. Of the 85 participants, 52 (61.2%) of them stated that at least one of their formal degrees (undergraduate, master's or PhD) have major computing content, 33 (38.8%) of them do not have such a background as seen in Figure *7*.

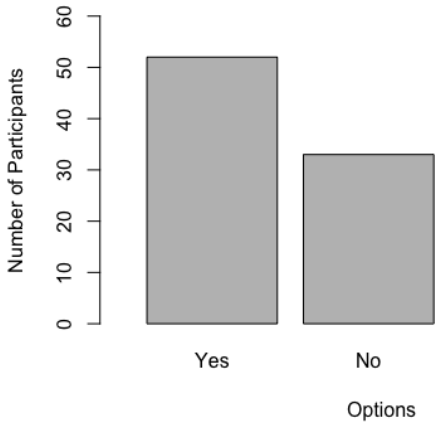


Figure 7. Having Formal Degree in Computing

4.1.4 Way of Obtaining Software Engineering and Software Development Skills

In this question (Q4), the participants were asked to rank the items from the highest ranking to the lowest ranking to state how they have obtained their software engineering and software development skills. Figure 8 shows the distribution of the highest ranked items. "Courses in Formal Education", "Attending Training Courses", "Co-workers", and "Learning on My Own" items were ranked the highest 26 (30.6%), 5 (5.9%), 11 (12.9%), and 41 (48.2%) times respectively. Two of the participants did not answer this question.

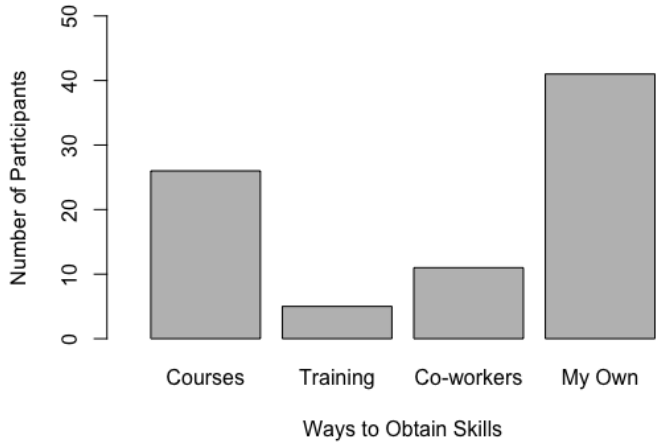


Figure 8. Ways to Obtain Software Engineering and Software Development Skills

4.1.5 Years of Developing Software

In this question (Q5), the participants were asked to state the number of years they have actively developed software. The options were 1 to 3 years, 3 to 10 years, and more than 10 years. There are 6 (7.1%), 19 (22.4%), and 60 (70.5%) responses respectively as seen in Figure 9.

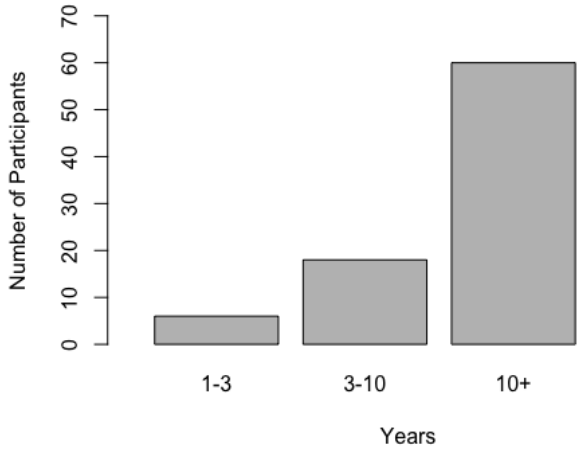


Figure 9. Years of Developing Software

4.1.6 Primary Job Description

In this question (Q6), the participants were asked to state their primary job description. Out of 85, 71 (83.5%) of the participants declared their job description as "researcher who does some software development", 2 (2.4%) of them declared as "software development", 12 (14.1%) of them declared as "software developer who does some research". Note that there is one more option as "software maintenance" (Option 3 in the question), but none of the participants chose that one as seen in Figure 10.

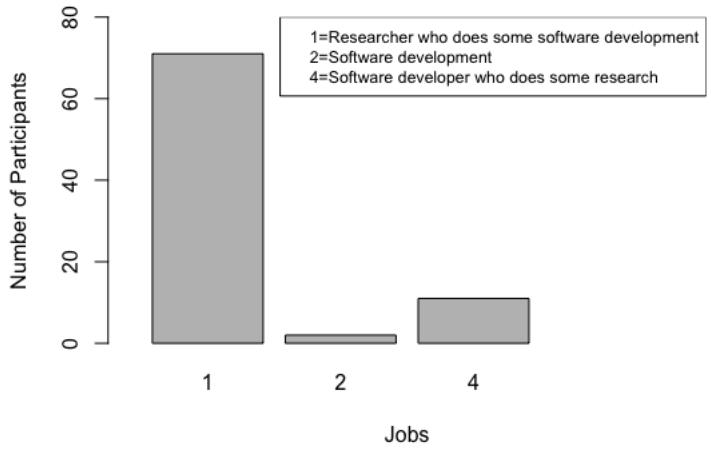


Figure 10. Primary Job Description

4.1.7 Development Team Sizes

In this question (Q7), the participants were asked to state how frequently (Never, Rarely, Sometimes, Often, Always) they are involved in different sized development teams (Single Person; Small (2 to 6 people) to Medium Team (7 to 12 people); and Large Team (more than 12 people) to Scientific Community). Originally, there were five separate categories which were grouped into three categories for easier statistical analyses. As shown in Figure 11, out of 85, 28 (32.9%) of the participants develop software by their own, 40 (47.1%) of the participants are either part of a small or medium team and 17 (20%) of the participants are either part of a large team or scientific community. Note that, those who indicated that they were more than one type of group were placed to the group type which they were involved with most frequently for ease of further statistical analysis. For example, if one participant indicated being rarely

involving in small to medium teams and often in large team to scientific community, this participant was placed in the latter group.

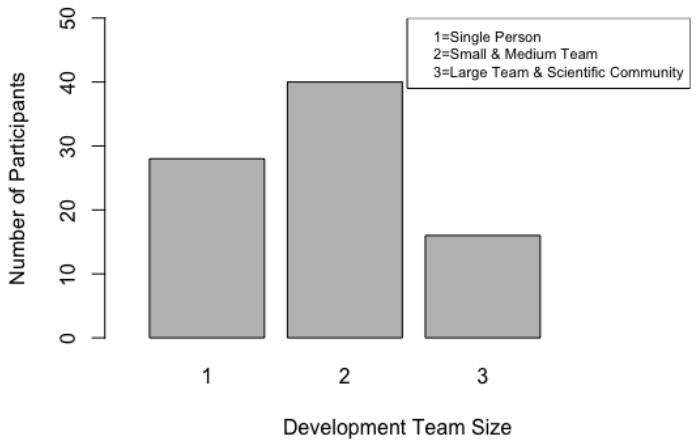


Figure 11. Development Team Sizes

4.1.8 Intended User Base Sizes

In this question (Q8), the participants were asked to state the frequency (Never, Rarely, Sometimes, Often, Always) of their intended user base size (Only the Developer to Small Local Group (less than 10 people); Larger Group (up to 100 people) to Scientific Community). Original separate four categories were regrouped into two for ease of further statistical analyses. As seen in Figure 12, the intended user base size of 37 (43.5%) participants are either for themselves or for small local group. The intended user base size of 48 (56.5%) participants is either larger group or scientific community. Note that, those who indicated that their user base size fits in more than one group were placed into the most frequent one as done with development team sizes as explained in Section 4.1.7.

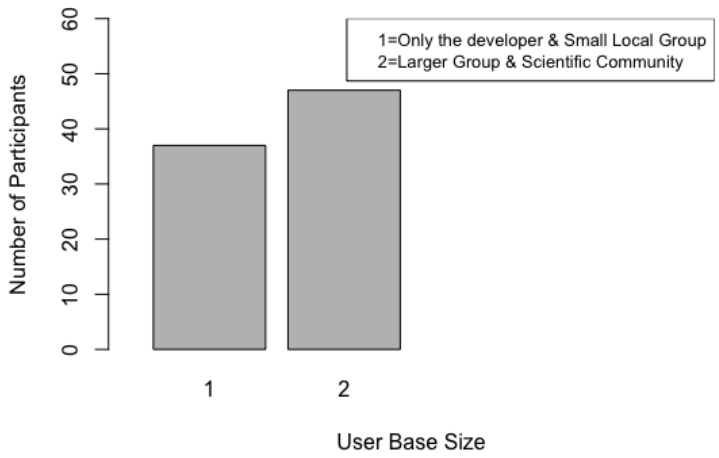


Figure 12. Intended User Base Sizes

4.1.9 General Software Engineering Practices

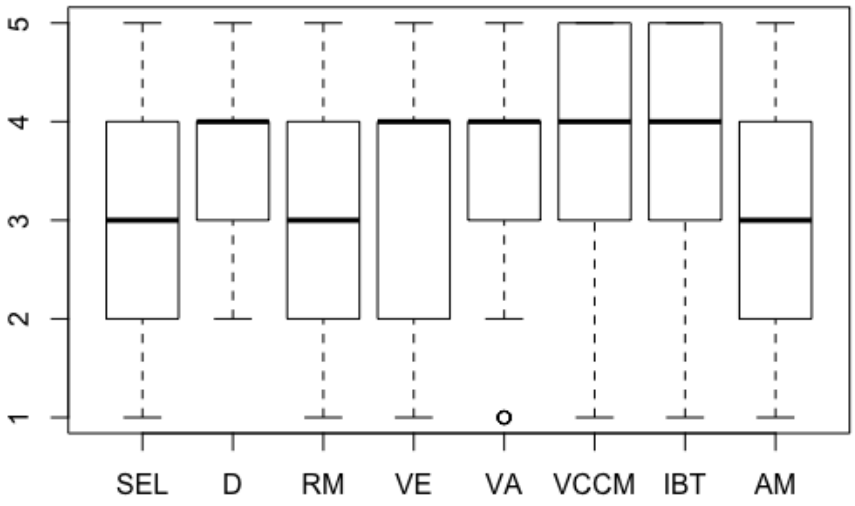
This was the second section of the survey containing 8 questions. The participants were expected to state their familiarity for eight of the software engineering practices through 5-point scale (1-None, 2-Low, 3-Medium, 4-High, 5-Very High). The definition of each practice was provided in case they were not known by the participants. The question format each of these eight questions can be exemplified with the following question for the software engineering lifecycles as shown in Figure 13. The medians and quartile distributions of each sub-part of the questions are shown in the figures in the order shown in the sample question. As can be seen in Figure 14, Figure 15, Figure 16, Figure 17, Figure 18 with the exception of version control and change management, participants find software engineering practices relevant but their familiarity and level of use is generally lower. In addition, personal familiarity and team familiarity distributions seem comparable but personal level of use tend to be higher than team level of use in software engineering lifecycles, documentation, and issue/bug tracking. The participants claim medium level familiarity in agile methods in both personally and at team level, but their level of use looks significantly lower both in personally and team level.

9) Software Engineering Lifecycles

sequence of activities that leads to the production of a software product.

	None	Low	Medium	High	Very High
Relevance to My Work	☐	☐	☐	☐	☐
Personal Familiarity	☐	☐	☐	☐	☐
Personal Level of Use	☐	☐	☐	☐	☐
Team Familiarity	☐	☐	☐	☐	☐
Team Level of Use	☐	☐	☐	☐	☐

Figure 13. Sample Question in Software Engineering Practice Part of the Survey



[16]

Figure 14. Median of the Practices for "Relevance To My Work" Option

[16] The common legend for Figures 14 to 18 is as follows: SEL-Software Engineering Life-cycles, D-Documentation, RM-Requirement Management, VE-Verification, VA-Validation, VCCM-Version Control/Change Management, IBT-Issue/Bug Tracking, AM-Agile Methods

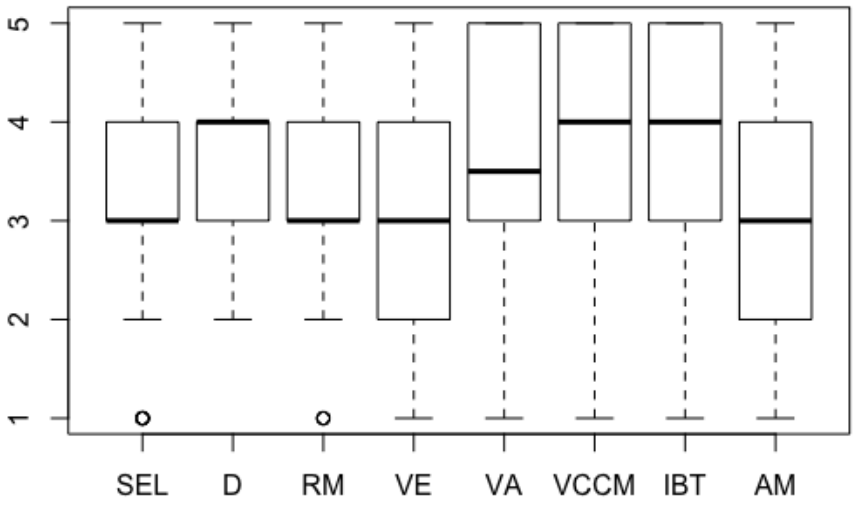


Figure 15. Median of the Practices for “Personal Familiarity” Option

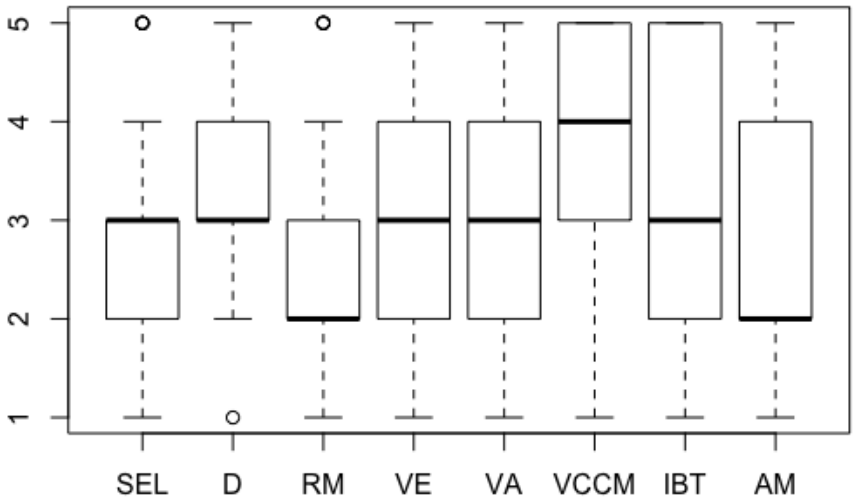


Figure 16. Median of the Practices for “Personal Level of Use” Option

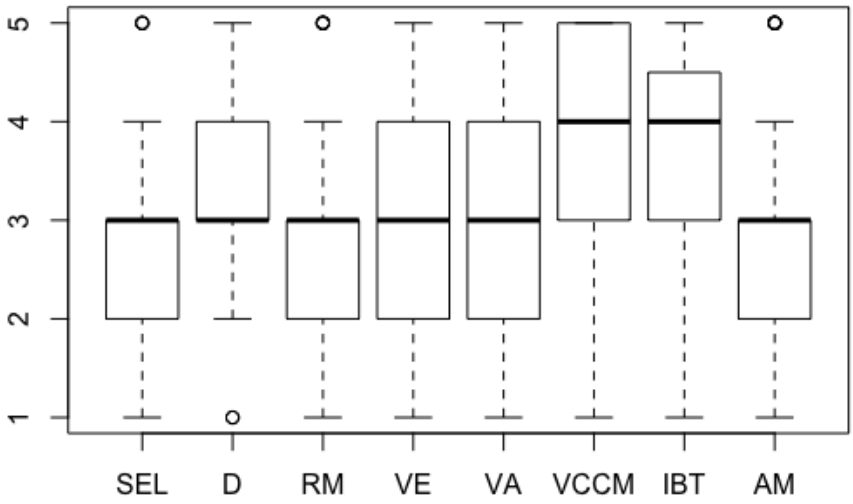


Figure 17. Median of the Practices for “Team Familiarity” Option

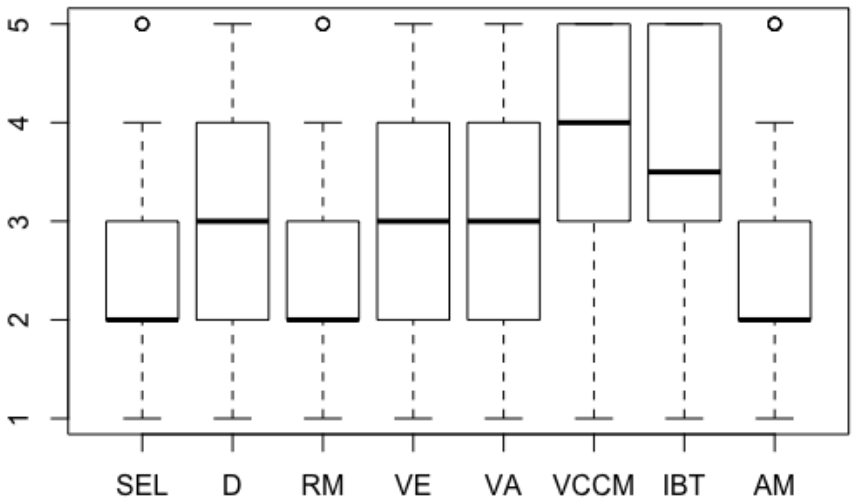


Figure 18. Median of the Practices for “Team Level of Use” Option

4.1.10 Scientific Software Development

The third section of the survey contained 9 questions that probed the opinions of the participants on various aspects of software and model development. The participants were expected to state their opinions for nine items through 6-point scale (0-No idea, 1-I strongly disagree, 2-I disagree, 3-Neutral, 4-I agree, 5-I strongly agree). The wording of the questions can be seen below. The median and quartile distributions of the items in scientific software development opinions section can be seen in Figure *19*. The minimum and maximum values are shown at the end of dashed line excluding outliers. Outliers are shown as small circles. The thick line that divides the box into two parts shows the median. The end of the box shows the upper and lower quartiles. The distribution of the answers shows that researchers in cognitive modelling do agree with issues observed to be problematic.

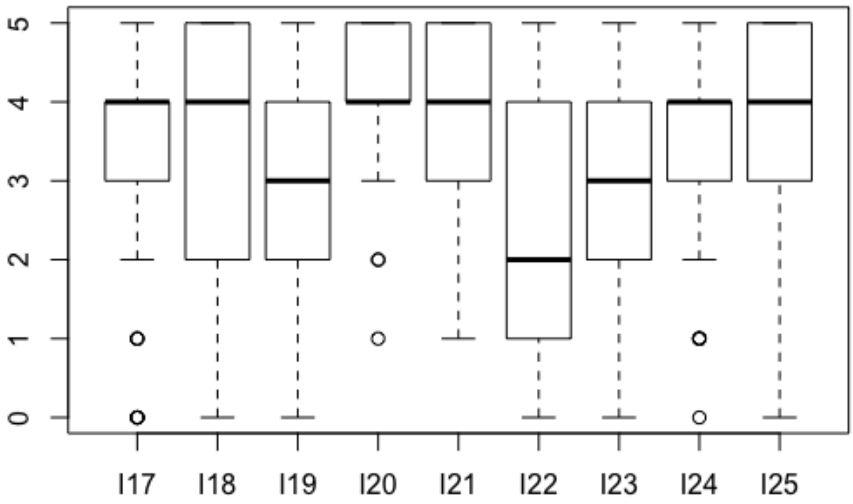


Figure 19. Median of the Items in Scientific Software Development Opinions Section

- I17: Frequent changes in software team members is a problem in cognitive/neuroscientific modeling software development.
- I18: Specifying requirements precisely in advance is not easy in cognitive/neuroscientific modeling software development.
- I19: Designing test cases is problematic in cognitive/neuroscientific modeling software development.

- 120: For sustainability of cognitive/neuroscientific modeling projects better documentation is necessary.
- I21: Performance efficiency is a concern in my cognitive/neuroscientific modeling software development practices.
- I22: Usability is not one of my primary concerns in my cognitive/neuroscientific modeling software development practices.
- I23: In cognitive/neuroscientific modeling, it is problematic to separate the theoretical model from the actual code.
- I24: In peer reviewing procedures for publishing, it is important to have the code reviewed as well as the manuscript.
- I25: Learning more about good software engineering practices would improve my software development work for cognitive/neuroscientific modeling.

4.2 Inferential Statistics

Since our data is ordinal and it is not normally distributed, non-parametric tests are appropriate to analyze it. Mann-Whitney U and Kruskal-Wallis tests are used to compare whether there is a difference in the dependent variables when there are 2 and 3 independent groups respectively. Also, chi-square test is used to decide whether there is a significant relationship between two variables. An alpha level of .05 is used in all statistical tests.

To recall, our hypotheses were following:

- Hypothesis 1) Similar problems exist in high level cognitive modelling/computational neuroscience and other scientific software development,
- Hypothesis 2) Developers comply with (are more familiar, find more relevant, and use more) software engineering practices more than the modellers,
- Hypothesis 3) Computational neuroscience framework developers and modellers comply with (are more familiar, find more relevant, and use more) software engineering practices more than high level cognitive modelling framework developers and modellers.

4.2.1 Tests of Hypothesis 1

In light of findings from the literature review of the scientific software development, major observations are formulated as sub-hypotheses below to compare them with the high-level cognitive modelling and computational neuroscience domains. The statistical results of tests with these hypotheses will be presented here and the results will be evaluated in Section 4.3.

In order to test Hypothesis 1, sub-hypotheses are created as follows:

- *Hypothesis 1.a:* The participants that have formal degree in computing content comply with software engineering practices more than the participants that do not have formal degree in computing content
- *Hypothesis 1.b:* When the development team size gets larger, more compliance with software engineering practices is observed
- *Hypothesis 1.c:* When the intended user base size gets larger, more compliance with software engineering practices is observed
- *Hypothesis 1.d:* Most scientists obtain their software development skills on their own or from their co-workers rather than through education.
- *Hypothesis 1.e:* Even though software development practices are thought to be relevant, some of the software engineering practices may significantly differ in their degree of relevance versus their personal level of use. For example, agile practices may be found relevant for a developer, but she is not using it personally to develop software. On the other hand, she may apply agile practices but she thinks that these practices are not relevant.

4.2.1.1 Tests of Hypothesis 1.a

Since there were 2 groups (the ones that have formal degree in computing content and the ones that do not) in Hypothesis 1.a, Mann-Whitney U test was performed. Each group was compared in both "Personal Level of Use" option in general software engineering practices and scientific software development opinions.

In general software engineering practices, there were no significantly difference of the practices except the "Software Engineering Lifecycles" between these groups. The test indicated that the ones that do not have formal degree in computing content (*Mdn* = 2.5), apply

software engineering life-cycles practices more than the ones that have formal degree in computing content (*Mdn* = 2), *W* = 1094.5, *p* = .001.

In scientific software development opinions, there were no significantly difference of the items except the "Specifying requirements precisely in advance is not easy in cognitive/neuroscientific modelling software development" and "Designing test cases is problematic in cognitive/neuroscientific modelling software development" between these groups. The test indicated that the ones that do not have formal degree in computing content (*Mdn* = 2.5), agree with the opinions more than the ones that have formal degree in computing content (*Mdn* = 2), *W* = 1099, *p* = .011; *W* = 1076, *p* = .002.

4.2.1.2 Tests of Hypothesis 1.b

Since there were 3 groups (single person, small – medium team, and large team – scientific community) in hypothesis 1.b, Kruskal-Wallis test was performed. Each group was compared in both "Personal Level of Use" option in general software engineering practices and scientific software development opinions. Because multiple analyses were performed, Bonferroni correction was applied. The p-value for Bonferroni correction can be found by dividing p-value, .05, by number of the comparisons made, 3. This revealed the new p-value for this hypothesis as .017.

In general software engineering practices, there were no significantly difference of the practices except the "Verification" and "Validation" between the groups. To find out which group cause the difference, Mann-Whitney U test was performed among the paired groups. The test indicated that participants in larger groups (*Mdn* = 4), apply verification and validation practices more than single person (*Mdn* = 3) and participants in smaller groups (*Mdn* = 3), *W* = 442.5, *p* = .001; *W* = 425, *p* = .007.

In scientific software development opinions, there were no significantly difference between the groups.

4.2.1.3 Tests of Hypothesis 1.c

Since there were 2 groups (only the developer – small local group and larger group – scientific community) in hypothesis 1.c, Mann-Whitney U test was performed. Each group was compared in both "Personal Level of Use" option in general software engineering practices and scientific software development opinions.

In general software engineering practices, there were no significantly difference of the practices except the "Issue/Bug Tracking" between these groups. The test indicated that participants of the software used by larger community (*Mdn* = 3), apply issue or bug tracking practice more than participants of the software used by smaller community (*Mdn* = 2), W = 506.5, p = .005.

In scientific software development opinions, there were no significantly difference of the items except the "Usability is not one of my primary concern in cognitive/neuroscientific modelling software development" between these groups. The test indicated that participants of the software used by larger community (*Mdn* = 3), agree with the previous opinion more than participants of the software used by smaller community (*Mdn* = 2), W = 506.5, p = .041.

4.2.1.4 Tests of Hypothesis 1.d

Since there were 2 groups (the ones that learn through education, courses in formal education – attending training courses and the ones that learn their own – from co-workers) in hypothesis 1.d, Mann-Whitney U test was performed. Each group was compared in both "Personal Level of Use" option in general software engineering practices and scientific software development opinions.

Both in general software engineering practice and scientific software development opinions parts, there were no significantly difference between the groups.

4.2.1.5 Tests of Hypothesis 1.e

A chi-square test was performed to examine the relation between software engineering practices of "Relevance to My Work" and "Personal Level of Use" options. The relation between these variables for each practice was significant as seen in Table *2*. The distribution of the results in Figure 14 and Figure 16 showed that although, participants found the practices relevant to their work, their personal level of use was lower except for software engineering lifecycles and version control.

Table 2. Chi-square Test Results of Relevance and Personal Level of Use of Practices

Practice	**Relevance and Personal Level of Use**
Software Engineering Life-cycles	$\chi^2(16,N=85)=84.25$, $p<.001$
Documentation	$\chi^2(12,N=85)=57.46$, $p<.001$
Requirement Management	$\chi^2(16,N=85)=84.15$, $p<.001$
Verification	$\chi^2(16,N=85)=122.32$, $p<.001$
Validation	$\chi^2(16,N=85)=102.02$, $p<.001$
Version Control/ Change Management	$\chi^2(16,N=85)=94.26$, $p<.001$
Issue/Bug Tracking	$\chi^2(16,N=85)=103.71$, $p<.001$
Agile Methods	$\chi^2(16,N=85)=156.16$, $p<.001$

4.2.2 Tests of Hypothesis 2

Since there were 2 groups (modellers and developers) in hypothesis 2, Mann-Whitney U test was performed to determine if the developers comply with software engineering practices more than the users. Each group was compared in both "Personal Level of Use" option in general software engineering practices and scientific software development opinions.

In general software engineering practices, there were no significantly difference of the practices except the "Validation" between these groups. The test indicated that modellers (*Mdn* = 3), apply validation practice more than developers (*Mdn* = 2), $W = 298$, $p = .025$.

In scientific software development opinions, there were no significantly difference of the items except the "Frequent changes in software team members is problematic in cognitive/neuroscientific modelling software development" between these groups. The test indicated that modellers (*Mdn* = 3), agree with the previous opinion more than developers (*Mdn* = 2), $W = 305$, $p = .009$.

4.2.3 Tests of Hypothesis 3

Since there were 2 groups (high level cognitive experts and computational neuroscientists) in hypothesis 3, Mann-Whitney U test was performed to determine if computational neuroscience framework developers and modellers comply with software

engineering practices more than high level cognitive modelling framework developers and modellers. Each group was compared in both "Personal Level of Use" option in general software engineering practices and scientific software development opinions.

In general software engineering practices, there were no significantly difference of the practices except the "Issue/Bug Tracking" between these groups. The test indicated that high level cognitive researchers (*Mdn* = 3), apply issue or bug tracking practice more than computational neuroscientists (*Mdn* = 2), $W = 560$, $p = .018$.

In scientific software development opinions, there were no significantly difference of the items except the "In peer reviewing procedures for publishing, it is important to have the code reviewed as well as the manuscript" between these groups. The test indicated that high level cognitive researchers (*Mdn* = 3), agree with the previous opinion more than computational neuroscientists (*Mdn* = 2), $W = 652$, $p = .031$.

4.3 Discussion

The response rate of the survey is 6.07%. Considering the time constraint and scope of the study, will be discussed in detail below, it is a fair percentage. The participants mostly work as researchers, majority as a developer, in computational neuroscience area, have a formal degree in computing content, learn necessary software development skills by themselves, develop software more than 10 years, involve in medium-size teams, and develop software for large user sizes.

4.3.1 Discussion of Individual Comments

The participants were allowed to add comments on the survey in the last section. There were 41 comments, that corresponds 48.24% of the participants, which include suggestions for the survey, state situations that currently faced, and state problems in the domain.

Two of the participants indicated problems regarding to readability of the survey. For one of them, questions are hard to understand. For the other one, it is not clear what is required for some questions. Also, there were some suggestions such as scaled questions could have "not-applicable" option, to include questions related to languages and development environments, to emphasize of UI/UX and visualization modules.

The difference between traditional and scientific software mentioned in three comments with such sentences "Scientific software is not a product", "The goals are clearly stated, and

procedures are clearly specified in traditional software", "The needs and goals are different between these software".

Two of the comments addressed budget issues in scientific software development. It is stated that applying software engineering practices takes lots of time and this situation costs more money.

Four of the participants stressed team member issues such as frequent changes within team members and team member diversity. One of them especially said that a current project lacks software engineering practices because the team consists of researchers whose main fields are different.

There are comments related to the shared code within the publications. Two of the participants stated that shared code should follow some standards (format, documentation) so that it can be downloaded and used easily. One participant reminded that some major publishers (Springer, Elsevier) offer special publication opportunities focused on the code.

Five participants strongly agreed on software engineering practices improved their work. One of them stated that even though he does not apply practices in personal projects, his team strictly applies software engineering practices. The other one indicated his code is not qualified well in software engineering practices, but he admires those researchers who have strong software engineering background.

4.3.2 General Discussion

Unlike previous surveys in scientific software, we asked participants whether they had formal degree with major computing or software engineering content or not. Surprisingly, the ones that do not have formal degree in computing content claim to apply (in the sense of personal level of use) software engineering life-cycle knowledge more than the ones that have formal degree in computing content. This might have been an indication of a need of a better probe to understand how exactly the researchers interpret applying software engineering lifecycles. Larger teams in cognitive modelling domain apply software engineering practices, verification and validation, more than smaller teams. Hannay et al. (2009) also hypothesized an increased team size affect perception for need for good software engineering practices, but they find the attributed importance was only and directly parallel with team size in software requirements and software maintenance. Our results indicated other software engineering areas can also be positively affected with increasing team sizes. Similarly, scientists who develop software for larger user size apply software engineering practices, issue/bug tracking, more than

the scientists who develop software for smaller user size. Hannay et al. (2009) also remarked that the scale of the software user base is either small or very large (more than 5000 users) - and this brings challenges in testing and debugging as developers do not have complete control of the software. The findings of the present thesis show that differences in intended user base can also imply developers make more use of some software engineering techniques. These findings support our hypothesis that similar problems exist in cognitive modelling and scientific software development. Another surprising result is that modellers apply some of the software engineering practices, for instance validation, more than the developers, which contradicts with our Hypothesis 2. We did have a relatively small sample of modellers as compared to developers and the term "validation" might have been more familiar in terms of philosophy of science to modellers than other software engineering related terminology. Therefore, this result should be reinterpreted within a better understanding of the overall survey. The other surprising result is that the high-level cognitive modelling experts apply some of the software engineering practices, for instance issue/bug tracking, more than the computational neuroscientists, which contradicts with our Hypothesis 3. The fact that cognitive architectures have been around for a longer time than computational neuroscience frameworks might have contributed to the result. The results can be better understood with complementary techniques such as structured interviews as they are based on subjective views of the participants.

Moreover, researchers in cognitive modelling stated their opinions about the software development literature. With this regard, surprisingly, they did not state that testing, and separating the theoretical model from the actual code were problematic. They emphasized that frequent changes in team members causes problems, specifying requirements is not easy, better documentation is necessary, and learning more about software engineering practices can be beneficial. Further, they also stated that trade-off between performance-efficiency is a concern, and reviewing the code is crucial in publishing. These findings are in line with the findings about problems of scientific software (Heaton and Carver, 2015; Storer, 2017; Johanson and Hasselbring, 2018).

4.3.3 Limitations

Although, 85 usable responses were collected, the results would be more accurate, if there were more participants. There were two reasons of not having more participants. First, the scope of the area in the study was not large, only high-level cognitive modelling and computational neuroscience. Second, we did not have much time to conduct this survey.

The validity of this survey could also be improved although we tried to obtain at least face-validity by conducting a pilot survey and adopting questions from previous surveys in scientific software development (Carver et al., 2013; Granados, 2000; Nguyen-Hoan et al., 2010)

We exclude mainly the software project management issues, e.g. problem of effective team management such as workload distribution or effective team leaders, scheduling (Granados, 2000). We exclude specific software engineering practices, such as design with UML, specific kinds of software testing such as regression testing, or other specific techniques such as refactoring. See (Nanthaamornphong and Carver, 2015; Granados, 2000) for examples of such surveys. We also exclude some demographic questions, not deemed relevant e.g. the type of organization participants works in or whether they produce production software as opposed to research software. See (Carver et al., 2013) for examples of such questions. Main reasons for excluding these questions were to be able to keep the survey under an acceptable length for topics that were not central to the research questions of this study. Hannay et al. (2009) also measured their participants' level of education and their current occupation. We did not distinguish between different degrees as in Hannay et al. (2009) or Carver et al. (2013) but we tried to collect whether any of the degrees were from computer science, software engineering or related fields. The downsize of this shortening was that we could not measure which degree influences the most software development skills as did Hannay et al. (2009). Instead of asking specifically various job titles as in (Hannay et al., 2009) we adopted the current occupation from (Granados, 2000) to distinguish researcher vs software developer perspective as in Q6. Hannay et al. (2009) distinguished between developing and using scientific software; we rather distinguished between developing and modelling where modelling is a specific subset of the using concept where users write mostly code for the models they develop.

CHAPTER 5

CONCLUSION

Using an online survey, this study was set out to evaluate the use of software engineering practices by cognitive modelling researchers. The data was collected from cognitive and neuroscientific modelling experts. Descriptive and inferential statistics were obtained from the survey submitted by 85 respondents.

We failed to find statistical evidence that the developers comply with software engineering practices more than the modellers. We also failed to find statistical evidence that the computational neuroscientists comply with software engineering practices more than the high cognitive modelling researchers. However, several of the sub-hypotheses were statistically found to be similar in cognitive modelling area as compared to other domains that were studied in the literature from scientific software development perspective. With this regard, when the team size and intended user base size increase, complying with the software engineering practices also increase in cognitive modelling. There is a strong relationship between degree of relevance for the software engineering practices and level of use for these practices personally.

Furthermore, results showed that although researchers in cognitive modelling area indicated software engineering practices are important for their work, they do not tend to apply the practices. Their personal usage of the practices was observed more than their team usage. The results also showed that researchers in cognitive modelling agreed with issues observed to be problematic in the scientific software development literature. Precisely, they found frequent changes in teams to be problematic, specifying requirements to be hard, acknowledged the need for documentation and wanted to improve their software engineering compliances. However, they did not generally find developing test cases problematic nor they had a problem with separating actual code from the theoretical model. They indicated performance-efficiency is an issue in cognitive modelling and agreed in general with the need to have code reviews in publishing.

The main contribution of this thesis is to investigate the findings of scientific software development literature in a previously unstudied scientific domain, namely cognitive modelling. This is the first survey in the area in which different roles and different subdomains were tried to be distinguished in terms of software engineering practices. Although there were no prevailing differences seen in most cases, these results could be confirmed or investigated deeper with using other complementary methods of empirical investigation.

Future Work

The same survey or an extended one can be carried out with more participants with better pretesting for validity and reliability. A semi-structured interview could be conducted on Skype with voluntary participants of this survey to explore further the individual results, especially those that relate to subdomains of cognitive science and differences between developers and modellers.

REFERENCES

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review, 111*(4), 1036-1060. doi: 10.1037/0033-295X.111.4.1036.

Benureau, F. C. Y., & Rougier, N. P. (2018). Re-run, repeat, reproduce, reuse, replicate: transforming code into scientific contributions. *Frontiers in Neuroinformatics, 11*, 69. doi: 10.3389/fninf.2017.00069.

Bican Can. "An Evaluation of Cognitive Modeling Tools". Dissertation, Middle East Technical University, 2008.

Carver, J., Heaton, D., Hochstein, L. & Bartlett, R. (2013). Self-perceptions about software engineering: a survey of scientists and engineers. *Computing in Science and Engineering, 15*(1), 7-11. doi: 10.1109/MCSE.2013.12.

Franklin, S., & Patterson, F. G. (2006). The Lida architecture: adding new modes of learning to an intelligent, autonomous, software agent. *Integrated Design and Process Technology, 703*, 764-1004.

Granados, A. F. (2000). A "Scientific" Approach to Software Project Management: Part II: Results of a Survey of Scientific Computing. *Astronomical Data Analysis Software and Systems IX, 216*, 20-23.

Hannay, J. E., Macleod, C., Singer, J., Langtangen, H. P., Pfahl, D., & Wilson, G. (2009). How do scientists develop and use scientific software?. *American Scientist*, 1-8. doi: 10.1109/SECSE.2009.5069155.

Heaton, D. "Software Engineering for Enabling Scientific Software Development". Dissertation, The University of Alabama, 2015.

Heaton D., & Carver, J. (2015). Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology, 67*, 207-219. doi: 10.1016/j.infsof.2015.07.011.

Johanson, A., & Hasselbring, W. (2018). Software engineering for computational science: past, present, future. *Computing in Science & Engineering, 20*(2), 90-109. doi: 10.1109/MCSE.2018.021651343.

Nanthaamornphong, A., & Carver, J. C., (2017). Test-driven development in scientific software: a survey. *Software Quality Journal, 25*(2), 343-372. doi: 10.1007/s11219-015-9292-4.

Nguyen-Hoan, L., Flint, S., & Sankaranarayana, R. (2010). A survey of scientific software development. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '10,* 12:1-12:10. doi: 10.1145/1852786.1852802.

Pflüger, D., Mehl, M., Valentin, J., Lindner, F., Pfander, D., Wagner, S., Graziotin, D., & Wang, Y. (2016). The scalability-efficiency/maintainability-portability trade-off in simulation software engineering: examples and a preliminary systematic literature review. *Proceedings of the Fourth International Workshop on Software Engineering for HPC in Computational Science and Engineering,* 19-27. doi: 10.1109/se-hpccse.2016.8.

Ramakrishnan, L., & Gunter, D. (2017, October). Ten Principles for Creating Usable Software for Science. *2017 IEEE 13th International Conference on e-Science (e-Science*), Auckland, New Zealand. doi: 10.1109/eScience.2017.34

Rosenbloom, P. S., Demski, A., & Ustun, V. (2016). The Sigma cognitive architecture and system: towards functionally elegant grand unification. *Journal of Artificial General Intelligence, 7*(1), 1-103. doi: 10.1515/jagi-2016-0001.

Say, B. (2008). Computational Models of Mind [PowerPoint slides]. Retrieved from http://ocw.metu.edu.tr/course/view.php?id=97.

Sommerville, I. (2011). *Software Engineering*. Boston: Pearson.

Storer, T. (2017). Bridging the chasm: a survey of software engineering practice in scientific programming. *ACM Computing Surveys, 50*(4), 1-32. doi: 10.1145/3084225.

Taatgen, N. A., Vugt, M. K. van, Borst, J. P., & Mehlhorn, K. (2015). Cognitive modeling at ICCM: state of the art and future directions. *Topics in Cognitive Science*. *8*(1), 259-263. doi: 10.1111/tops.12185.

Tikidji-Hamburyan, R. A., Narayana, V., Bozkus, Z., & El-Ghazawi, T. A. (2017). Software for brain network simulations: a comparative study. *Frontiers in Neuroinformatics, 50*, 1-16. doi: 10.3389/fninf.2017.00046.

Trappenberg, T. P. (2010). *Fundamentals of Computational Neuroscience*. Oxford: Oxford University Press.

Wagner S., Pflüger, D., & Mehl, M. (2015). Simulation software engineering: experiences and challenges. *Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering - SE-HPCCSE '15*, 1-4. doi: 10.1145/2830168.2830171.

APPENDICES

APPENDIX A
QUESTIONNAIRE FORM

This is the whole survey in word format. Because of LimeSurvey constraints, some items are slightly different visually in the online survey published at http://moodle.atilim.edu.tr/anket/index.php?r=survey/index&sid=579941&lang=en

Thanks for your interest in participating our survey on software engineering practices in cognitive modeling and computational neuroscience. This study is conducted as a part of Furkan Kurtaran's Master Thesis under the supervision of Dr. Bilge Say and you can contact both of the researchers for more information. You will be asked a series of questions on your computing background, current software development environment, and your opinions about software engineering practices in cognitive/neuroscientific modeling. The survey will take about 10 minutes to complete. We hereby confirm that all data will be treated anonymously. You can leave the survey at any page. As a symbol of gratitude to our participants, we will offer a gift check from Amazon to one of our participants. You will be informed of the details at the end of the survey.

Q4, Q9 to Q16 are adopted and revised from (Carver et al., 2013); Q5, Q7, Q8 from (Nguyen-Hoan et al., 2010); Q6 from (Granados, 2000).

With Regards,

Furkan Kurtaran, Master's Student
Department of Computer Engineering, Atilim University, Turkey
furkan.kurtaran@atilim.edu.tr
https://www.atilim.edu.tr/en/compe/page/1595/academic-staff

Dr. Bilge Say, Faculty Member

Department of Software Engineering, Atilim University, Turkey

bilge.say@atilim.edu.tr

https://www.atilim.edu.tr/en/se/page/2286/academic-staff

References:

Carver, J., Heaton, D., Hochstein, L. & Bartlett, R. (2013). Self-perceptions about software engineering: a survey of scientists and engineers. *Computing in Science and Engineering, 15*(1), 7-11. doi: 10.1109/MCSE.2013.12.

Nguyen-Hoan, L., Flint, S., & Sankaranarayana, R. (2010). A survey of scientific software development. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '10,* 12:1-12:10. doi: 10.1145/1852786.1852802.

Granados, A. F. (2000). A "Scientific" Approach to Software Project Management: Part II: Results of a Survey of Scientific Computing. *Astronomical Data Analysis Software and Systems IX, 216,* 20-23.

Please be aware that once you click the "next" button, your answers are saved and you cannot come back to previous pages. You will be asked to "submit" at the end of the survey. If you have to leave the survey unintentionally before submitting, please start over.

Demographics

1) How would you describe your software development experience in cognitive science/neuroscience? (Choose the column that applies most. Mark all that apply within the column.)

For all other questions, “the software” will imply to your choices in question 1&2. “The software” can imply both an architecture/framework/simulator itself and a

computational model that is written in a kind of programming language on such a platform.

"Model developers" are also referred to as "software developers" in this survey.

	High Level Cognitive Modeling	Computational Neuroscience
Developer of a Core Architecture/Framework/Simulator	☐	☐
Developer of Extensions/Optional Modules of a Architecture/Framework/Simulator	☐	☐
Model Developer	☐	☐

2) Name the main cognitive architecture/computational neuroscience framework/cognitive modeling tool you have developed/used

. . . .

3) Are any of your formal degrees (undergraduate/master's/PhD) in one of the following majors:

Software Engineering
Computer Science
Computer Engineering
Information Systems
Other major, similar to above degrees in terms of computing content

☐Yes / ☐ No

4) How have you obtained your software engineering and software development skills?

Double-click or drag-and-drop items in the left list to move them to the right – your highest ranking item should be on the top right, moving through to your lowest ranking item.

Courses in Formal Education
Attending Training Courses
Co-workers
Learned on My Own

5) How many years have you been actively developing software?

☐ 1 – 3
☐ 3 – 10
☐ 10+

6) Would you consider your primary job description to be

☐ Researcher who does some software development
☐ Software development
☐ Software maintenance
☐ Software developer who does some research

7) What is your software development team size?

	Never	Rarely	Sometimes	Often	Always
Single Person	☐	☐	☐	☐	☐
Small Team(2-6)	☐	☐	☐	☐	☐
Medium Team (7-12)	☐	☐	☐	☐	☐
Large Team (12+)	☐	☐	☐	☐	☐
Scientific Community	☐	☐	☐	☐	☐

8) The intended user base size for the software I develop is

	always	often	sometimes	rarely	never
only the developer (myself)	☐	☐	☐	☐	☐

small local group (less than 10)	☐	☐	☐	☐	☐
larger group (up to 100)	☐	☐	☐	☐	☐
scientific community	☐	☐	☐	☐	☐

General Software Engineering Practices

For each of the following activities, please indicate your familiarity, your level of use, your team's familiarity, and your team's level of use.

- Definitions are mostly quotes from (Sommerville, 2011), provided for those who are not familiar with software engineering terminology.
- If you do not develop software as a team, you can leave team related lines empty.

9) Software Engineering Lifecycles

sequence of activities that leads to the production of a software product.

	None	Low	Medium	High	Very High
Relevance to My Work	☐	☐	☐	☐	☐
Personal Familiarity	☐	☐	☐	☐	☐
Personal Level of Use	☐	☐	☐	☐	☐
Team Familiarity	☐	☐	☐	☐	☐
Team Level of Use	☐	☐	☐	☐	☐

10) Documentation

A tangible way of describing the different representations of a software system (requirements, UML, code, etc.) and its production process

	None	Low	Medium	High	Very High
Relevance to My Work	☐	☐	☐	☐	☐
Personal Familiarity	☐	☐	☐	☐	☐
Personal Level of Use	☐	☐	☐	☐	☐
Team Familiarity	☐	☐	☐	☐	☐
Team Level of Use	☐	☐	☐	☐	☐

11) Requirements Management

process of writing down the user and system requirements in a requirements document

	None	Low	Medium	High	Very High
Relevance to My Work	☐	☐	☐	☐	☐
Personal Familiarity	☐	☐	☐	☐	☐
Personal Level of Use	☐	☐	☐	☐	☐
Team Familiarity	☐	☐	☐	☐	☐
Team Level of Use	☐	☐	☐	☐	☐

12) Verification

intended to show that a system conforms to its specification

	None	Low	Medium	High	Very High
Relevance to My Work	☐	☐	☐	☐	☐
Personal Familiarity	☐	☐	☐	☐	☐
Personal Level of Use	☐	☐	☐	☐	☐
Team Familiarity	☐	☐	☐	☐	☐
Team Level of Use	☐	☐	☐	☐	☐

13) Validation

intended to show that a system meets the expectations of the system customers (i.e researchers for modeling software)

	None	Low	Medium	High	Very High
Relevance to My Work	☐	☐	☐	☐	☐
Personal Familiarity	☐	☐	☐	☐	☐
Personal Level of Use	☐	☐	☐	☐	☐
Team Familiarity	☐	☐	☐	☐	☐
Team Level of Use	☐	☐	☐	☐	☐

14) Version Control / Change Management

Keeping track of the different versions of software components / proper tracking and control of any kind of change such as addition of new features to the software

	None	Low	Medium	High	Very High
Relevance to My Work	☐	☐	☐	☐	☐
Personal Familiarity	☐	☐	☐	☐	☐
Personal Level of Use	☐	☐	☐	☐	☐
Team Familiarity	☐	☐	☐	☐	☐
Team Level of Use	☐	☐	☐	☐	☐

15) Issue / Bug Tracking

Tracking the status and resolution of detected errors in the software as well as any issues to be resolved

	None	Low	Medium	High	Very High
Relevance to My Work	☐	☐	☐	☐	☐
Personal Familiarity	☐	☐	☐	☐	☐
Personal Level of Use	☐	☐	☐	☐	☐
Team Familiarity	☐	☐	☐	☐	☐
Team Level of Use	☐	☐	☐	☐	☐

16) Agile Methods

incremental development methods in which the increments are small and, typically, new releases of the system are created frequently

	None	Low	Medium	High	Very High
Relevance to My Work	☐	☐	☐	☐	☐
Personal Familiarity	☐	☐	☐	☐	☐
Personal Level of Use	☐	☐	☐	☐	☐
Team Familiarity	☐	☐	☐	☐	☐
Team Level of Use	☐	☐	☐	☐	☐

Scientific Software Development Opinions

Indicate your opinions for the following sentences.

17) Frequent changes in software team members is a problem in cognitive / neuroscientific modeling software development.

☐ I strongly agree ☐ I agree ☐ Neutral ☐ I disagree ☐ I strongly disagree ☐No idea

18) Specifying requirements precisely in advance is not easy in cognitive / neuroscientific modeling software development.

☐ I strongly agree ☐ I agree ☐ Neutral ☐ I disagree ☐ I strongly disagree ☐No idea

19) Designing test cases is problematic in cognitive / neuroscientific modeling software development.

☐ I strongly agree ☐ I agree ☐ Neutral ☐ I disagree ☐ I strongly disagree ☐No idea

20) For sustainability of cognitive/neuroscientific modeling projects better documentation is necessary.

☐ I strongly agree ☐ I agree ☐ Neutral ☐ I disagree ☐ I strongly disagree ☐No idea

21) Performance efficiency is a concern in my cognitive / neuroscientific modeling software development practices.

☐ I strongly agree ☐ I agree ☐ Neutral ☐ I disagree ☐ I strongly disagree ☐No idea

22) Usability is not one of my primary concerns in my cognitive / neuroscientific modeling software development practices.

☐ I strongly agree ☐ I agree ☐ Neutral ☐ I disagree ☐ I strongly disagree ☐No idea

23) In cognitive/neuroscientific modeling, it is problematic to separate the theoretical model from the actual code.

☐ I strongly agree ☐ I agree ☐ Neutral ☐ I disagree ☐ I strongly disagree ☐No idea

24) In peer reviewing procedures for publishing, it is important to have the code reviewed as well as the manuscript.

☐ I strongly agree ☐ I agree ☐ Neutral ☐ I disagree ☐ I strongly disagree ☐No idea

25) Learning more about good software engineering practices would improve my software development work for cognitive/neuroscientific modeling.

☐ I strongly agree ☐ I agree ☐ Neutral ☐ I disagree ☐ I strongly disagree ☐No idea

Feedback

26) We would appreciate if you would like to provide any comments on the above questions or in general on software engineering practices in cognitive modeling or computational neuroscience.

Thanks for participating in our survey. If you would like to receive more information on the results of this survey, or if you would like to participate further in our studies on software engineering practices in cognitive/neuroscientific modeling, please fill in the relevant parts below. As a gratitude for you participation, we would like to make a draw for a $40 amazon gift certificate to one of our participants. If you would like your name to be in the draw please fill the relevant part below.

Furkan Kurtaran, Master's Student
Department of Computer Engineering, Atilim University, Turkey
furkan.kurtaran@atilim.edu.tr
https://www.atilim.edu.tr/en/compe/page/1595/academic-staff

Dr. Bilge Say, Faculty Member
Department of Software Engineering, Atilim University, Turkey
bilge.say@atilim.edu.tr
https://www.atilim.edu.tr/en/se/page/2286/academic-staff

☐ Yes, I would like to receive more information about this survey's results
☐ Yes, I would like to participate in similar further studies.
☐ Yes, I would like to participate in the prize draw.

Name:
Institution:
E-mail:

APPENDIX B

INVITATION MESSAGE FOR THE QUESTIONNAIRE

Dear (Name)[17],

[Apologies for any duplicate receipt of this invitation]

As a part of Master's thesis research, I am conducting an online survey on software engineering practices and problems in developing cognitive and neuroscientific architectures/frameworks/simulators and computational models on those platforms. If you have been involved in developing such cognitive/neuroscientific modelling tools and computational models on them, we will be grateful if you could answer our survey questions in the link given below:

http://moodle.atilim.edu.tr/anket/index.php?r=survey/index&sid=579941&lang=en

The participation is voluntary and anonymous; the survey will take about 10 minutes. We hope to better understand and develop suggestions for improving cognitive modelling and computational neuroscience software from a software engineering point of view at the end of our research.

As a symbol of gratitude of your participation, participants who complete the survey will be eligible for a prize draw for a $40 book check from Amazon.

Best Regards,

Furkan Kurtaran

Computer Engineering Department

Research Assistant

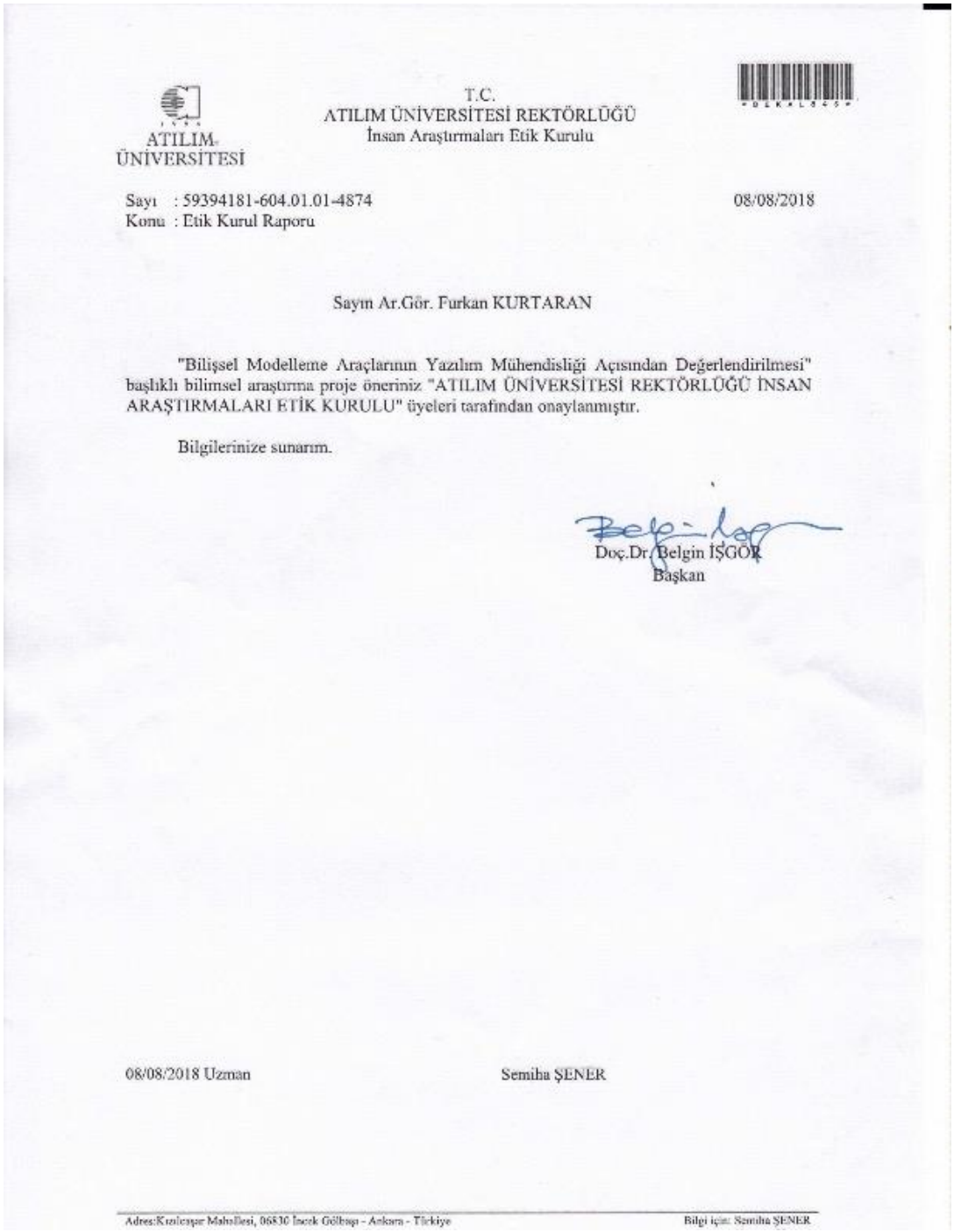
Atilim University

https://www.atilim.edu.tr/en/compe/page/1595/academic-staff

[17] Name of the individual

APPENDIX C

ETHICS COMMITTEE APPROVAL



ATILIM ÜNİVERSİTESİ

T.C.
ATILIM ÜNİVERSİTESİ REKTÖRLÜĞÜ
İnsan Araştırmaları Etik Kurulu

Sayı : 59394181-604.01.01-4874
Konu : Etik Kurul Raporu

08/08/2018

Sayın Ar.Gör. Furkan KURTARAN

"Bilişsel Modelleme Araçlarının Yazılım Mühendisliği Açısından Değerlendirilmesi" başlıklı bilimsel araştırma proje öneriniz "ATILIM ÜNİVERSİTESİ REKTÖRLÜĞÜ İNSAN ARAŞTIRMALARI ETİK KURULU" üyeleri tarafından onaylanmıştır.

Bilgilerinize sunarım.

Doç.Dr. Belgin İŞGÖR
Başkan

08/08/2018 Uzman

Semiha ŞENER

Adres:Kızılcaşar Mahallesi, 06830 İncek Gölbaşı - Ankara - Türkiye

Bilgi için: Semiha ŞENER