Thesis Project Portfolio

Developer Efficiency: How Operational Tools Streamline Software Engineering

(Technical Report)

Closing the Gap Between Engineering Education and Engineering Practice: An Anaylsis with Respect to Computer Science & How Trade Schools Could Be Looked to for the Solution

(STS Research Paper)

An Undergraduate Thesis

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Rohit Batra

Spring, 2022

Department of Computer Science

Table of Contents

Sociotechnical Synthesis

An Analysis of Computer Science Education and its Applicability in the Workplace

The STS topic I chose to explore and the technical project are related in the sense that the STS topic dives into the educational aspect of what an average computer science student experiences during their time at university while the technical project dives into what that same student would experience in a professional workplace environment. The STS topic is a discussion of the gaps between what is expected from professional engineers and what is taught to engineering students in university with a specific focus on computer science. The technical topic discusses a specific project experience at a large software company (Amazon) which showcases how the skills which are learned in a computer science degree program are applied in the workplace and the impact which they can have when applied correctly. Although the projects are not directly related, they both discuss relevant aspects of the pursuit of a computer science degree with the intention of going into industry.

In my STS research, I researched and discussed the gap between a typical engineering education and what is expected of engineers in the workplace and how to address this gap. My research focused specifically on Computer Science and the degrees translation to the work of a software engineer as one of the fields which a computer science enters upon graduating. The research discussed this in two specific aspects: what is the gap exactly and what specific actions can be taken to close it. The result of my research yielded concrete suggestions which could be implemented in the common engineering education to better prepare engineering students to enter the workplace.

The technical portion of my thesis discussed a specific project experience and goes into how software is developed at a large company, how software is debugged at a large company,

and how software developers have impact at a large company. Specifically, it discusses an operational tool which was developed as an intern project with the intention of improving developer efficiency. It goes into depth on how the tool was developed, and how it would save the teams' developers hours of time debugging, therefore streamlining their software engineering pipeline.

Completing both aspects of the thesis research simultaneously allowed me to learn a lot about what it means to get a computer science education at an engineering school and how it translates into industry. The STS research portion of the thesis allowed me to gain a better understanding of how computer science curriculums attempt to adequately prepare students to become engineers who practice engineering with ethics in mind, whereas the technical portion of the thesis allowed for me to reflect on an experience in which I acted as an engineer with autonomy and look at the implications of what I had learned in said curriculum. The thesis as a whole showcases not only the ethical and social responsibility which an engineer has as a student but also the ethical and social responsibility which they carry into the workplace as they enter industry.

Developer Efficiency: How Operational Tools Streamline Software Engineering

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Rohit Batra

Spring, 2022

Technical Project Team Members

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Daniel Graham, Department of Computer Science

Developer Efficiency: How Operational Tools Streamline Software Engineering

CS 4991 Capstone Report, 2022

Rohit Batra
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
rb4jx@virginia.edu

Abstract
Developers are expensive, so companies need their time used in the most efficient manner possible. A constant problem in the field of software engineering is making the process of software engineering as efficient as possible for the developers. At Amazon, I utilized the agile method, Python, and some proprietary AWS tools to develop a solution for streamlining the debugging process for developers. The outcome of my work was a useful tool which turned a manual process into an automated one and allowed the developers on my team to save significant amounts of time debugging tickets and even debug multiple tickets simultaneously. Furthermore, the tool developed could be expanded on for future use cases such as adding additional ticket types or expanding the breadth or depth of information which it retrieves.

1 Introduction

Debugging is a time consuming and challenging process which requires serious time and effort from developers. It is often a process with no clear timeline. Developers are forced to spend countless hours in search of the bug with no end in sight. Developers hate debugging, and companies hate it because it wastes the time of their expensive developers. Debugging is a process companies attempt to streamline to make it as efficient as possible. This was my task at Amazon. My team would receive tickets from the customers of the AWS Certificate Authorities and have to debug them. My task was to automate this process.

2 Related Works

Manifesto for Agile Software Development. [Online]. Available:
The Agile Manifesto (Manifesto) contains a brief description of the ideology behind the Agile development process. This was the process used for development at Amazon.

Metz (2015) discusses the foundation and success of GitHub – arguably the most useful tool for software engineers today. This showcases what type of tools are useful for software engineers and how they contribute to making the profession more efficient. It also demonstrates process and product similar to my project at Amazon.

3 Project Design

The process prior to my tools was something like this: Ticket comes in from customer -> developer looks at ticket-> developer manually gathers relevant information and finds relevant code -> developer searches relevant code for some time -> developer finds problematic code and patches it -> developer marks ticket as solved. The set of tools which I developed streamlined this process and now it looks more like this: Ticket comes in from customer -> developer looks at ticket-> developer runs automation tools with ticket number -> tools deliver all relevant information about this customer, their accounts, and the services which they are running to the developer -> developer searches relevant code for some time -> developer finds problematic code and patches it -> developer marks ticket as solved. It is largely still a similar process; however, one of the manual steps

which involved hours of developer labor was eliminated resulting in a net increase in productivity for the team.

4. Results

The newly-designed tool allowed for a workflow which normally took hours of tedious work to be completed in a matter of minutes and without the attention or focus of the developer. Moreover, this had a twofold effect as it allowed the developers to focus on more pressing issues while the tool would do the debugging automatically; they simply have to look at the output when the tool was finished.

5. Conclusion

Although only one set of tools was developed for one specific team of a large corporation over the course of my internship, the work which I did and the groundwork which was laid down will serve as the foundation for future developer tools for not only my team but other teams at Amazon as well. In the long run, improving developer efficiency will save the company millions of dollars as well as saving the developers many headaches and hours of frustration. This also showcases that not all work at a company is always about the customer. Making sure that the employees are satisfied is highly important as well.

6. Future Work

Using the foundation built, the team will continue to develop more Python based tools to continue improving developer efficiency. Specifically, my team will continue adding more ticket types to allow for the toolset to diagnose and debug additional types of customer issues efficiently. In a broader context, the Python foundation built could be shared with other teams at Amazon to allow them to improve their debugging processes in similar ways.

References

C. Metz, “How github conquered Google, Microsoft, and everyone else,” Wired, 12-Mar-2015. [Online]. Available: https://www.wired.com/2015/03/github-conquered-google-microsoft-everyone-else/. [Accessed: 14-Mar-2022].

Manifesto for Agile Software Development. [Online]. Available: https://agilemanifesto.org/iso/en/manifesto.html. [Accessed: 03-Apr-2022].

Closing the Gap Between Engineering Education and Engineering Practice: An Analysis with Respect to Computer Science & How Trade Schools Could Be Looked to for the Solution

A Research Paper submitted to the Department of Engineering and Society

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Rohit Batra

Spring 2022

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Kathryn A. Neeley, Associate Professor of STS, Department of Engineering and Society

STS Research Paper

Software engineering as a profession has been growing exponentially in the past few decades with the advent and rise of consumer technology. Specifically, the Bureau of Labor Statistics predicts a 22% growth rate in the industry from 2019 to 2029 which is much greater than the national average for all occupations of 4% ("Career and salary of Software Engineer" ComputerScience.org). The products of this growing field affect the day-to-day lives of almost every individual in the world. Unfortunately, software engineering curricula currently in place were adopted at a time when many industries rooted in computer science such as social media did not exist.

The destabilizing condition which makes this issue so important at this time is increasing use of commercial software along with the lack of sufficiently educated engineers resulting in inefficient, insecure, and insufficient software products. The research backing this up is clear and abundant.

Schilling points this out specifically in her research with a statement on how the traditional lecture format used in the computer science curriculums is inadequate. She even cites prior research done from 1996, further indicating that this is a destabilizing condition; it has been more than two decades, yet the educational format is still an issue and does not produce sufficiently talented engineers. "While some progress has been achieved, computer science (CS) teaching is still mainly based on the traditional lecture format (Van Gorp and Grissom 2001). Renkl, Mandl, and Gruber (1996) have pointed out that this educational approach often results in 'inert' knowledge that is not used for problem solving and concrete professional practice." (Schilling and Klamma, 367)

Moreover, Garousi discusses the addition of software engineering programs as programs separate from computer science with the intention to prepare students for industry. However, he discusses how these programs are still inadequate and exhibit the same disparity between traditional computer science curriculums and how they map to industry. “Many software engineering (SE) university programs have evolved from computer science programs and still focus on theoretical and technical computer science topics as well as mathematical foundations. This emphasis seems to cause a discrepancy between the skills learned from an SE university education and those needed in SE employment. In the community, some believe that ‘The software engineering shortage is not a lack of individuals calling themselves ‘engineers,’ the shortage is one of quality—a lack of well-studied, experienced engineers with a formal and deep understanding of software engineering.’” (Garousi, 68)

Both of these articles of research describe the clearly present disparity and the issues it causes in industry. Another example of how this disparity affects the industry can be seen very clearly in the case of unethical artificial intelligence and machine learning systems. A sufficient supply of ethical, talented, and properly-educated engineers who methodically develop software could have prevented many of the issues with artificial intelligence and machine learning and also other software systems. Though, this is just one specific example with respect to computer science.

This disparity has the ability to affect every engineering field and the daily lives of every person who utilizes technology if it is allowed to propagate. It is a fundamental issue about engineering education – “To continue to adequately serve the stakeholders of engineering education, it is imperative that engineering programs evolve. Educators must take a look not only at what is being taught, but also at how it is being taught. It is clearly demonstrated by student’s

perception of having sound professional design skills, contrasted with the evident unfulfilled needs and wants of industry that there is a significant disconnect between stakeholders. A structured program to enhance learning in the identified areas of need must be implemented in order to achieve acceptable outcomes in engineering education." (May and Strong)

Based off of the research done on the current state of the computer science curriculum and software engineering industry, a new understanding has been developed which encourages the undergraduate computer science degree to more closely align with industry expectations – at least Bachelor of Science degrees. Specifically, the engineering school at UVA has the core mission of "Our mission is to make the world a better place by creating and disseminating knowledge and by preparing engineering leaders to solve global challenges." (Our Mission) This mission would be better achieved by aligning the degree program in the engineering school to equip students to enter industry as opposed to preparing them for careers in academia.

Based on this new understanding, measures to shorten the gap between an engineering education and industry requirements should be taken, and engineering education should be reevaluated with its differences from a traditional liberal arts education and curriculum being brought to the forefront as elements which should be changed it.

What is the Gap: disparity between Engineering Education & Engineering Practice with respect to Computer Science

Over the years as the engineering industry has experienced rapid growth, the disparity between what engineers need to know for the job and what engineers come out of school knowing has become increasingly apparent. Conde discusses this in depth in his paper regarding bridging the gap between academia and industry. "Recent graduates often fail to meet industry expectations when they first enter the workforce." (Conde) "Although many courses in computer

science and software engineering require students to work on practical assignments, these are usually toy projects that do not come close to real professional developments." (Conde) Both of these quotes from the paper illustrate how apparent the disparity is currently. This portion of the paper will discuss this issue in specific as it relates to the disparity between a traditional Computer Science education and what the Software Engineering industry demands of Computer Science graduates who enter industry.

One of the main differences which I experienced personally during an internship is the focus on theory in the educational environment and the lack of adequate exposure to applicable workplace practices. There are a plethora of courses available on theoretical concepts, however, if a student wants to learn about relevant and modern day software engineering practices, there are only a handful of courses available at most universities if any. This needs to change in the computer science curriculum, and in other engineering curriculums in which there could be a similar lack of focus on industry practices.

Another key difference between a typical computer science curriculum and software engineering is the speed and manner in which projects are completed. In the academic environment, projects or assignments typically have one strict deadline and must be submitted in full at that deadline. This differs from the common practice in industry where things are done in an Agile way. The Agile methodology is a practice which was put into place in the software engineering industry in which software is developed in a periodical and cyclic way.

Figure 1. Illustration & description of Agile methodology which is currently the industry standard of software development (What is agile methodology?).

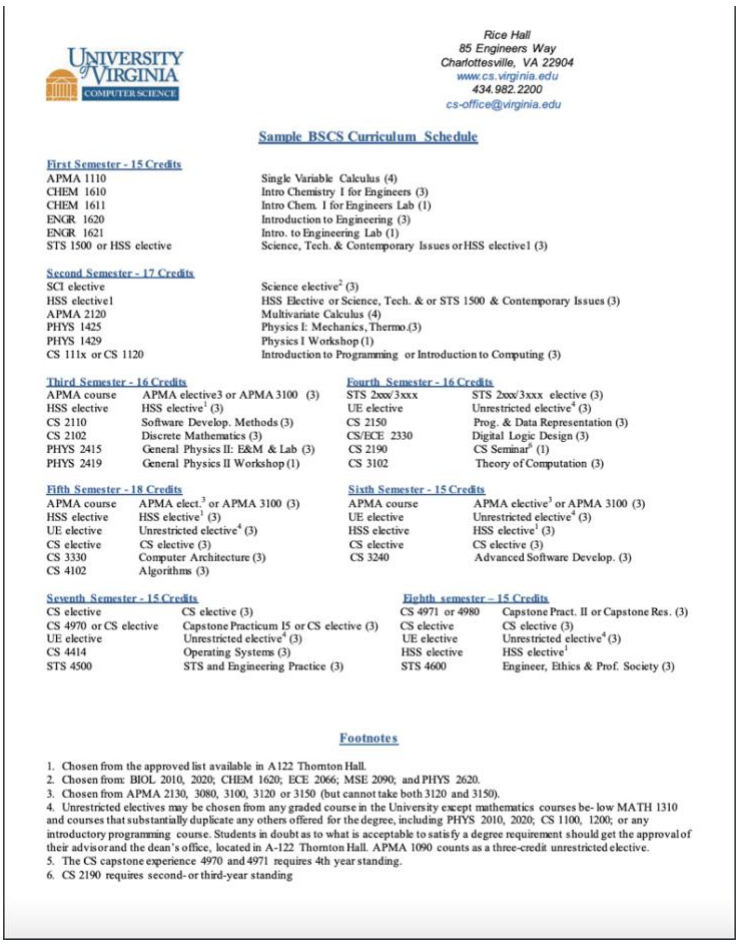
Here is a brief description of the Agile methodology by the software development consulting firm, Nvisia.

> Agile is not as much a methodology as it is a set of values, ideals and goals. Scrum and kanban are implementations of agile. Both practices are implemented differently while having the same foundation, the Agile Manifesto. They have specific, measurable and quantifiable procedures and processes. Both share the goal of breaking down projects into smaller chunks and focusing on continuous testing. The number one performance gain of agile is preventing rework. Applying either process in your company will help you spot deficiencies sooner and allow you the clarity to fix any problems that arise during the process, rather than at the end of the project. Being most effective, while adding business value, should be the end goal.

This is one example of what engineering graduates are forced to learn on the job at a rapid pace with high stress with the current curriculums in place. Situations like this can be easily avoided with structured changes to the curriculum and the addition of courses which teach students about industry practices. Making it so that young graduates are forced to enter the workplace and quickly learn the industry standards in high pressure situations is simply not a good idea when it can be easily avoided.

This also relates to the previous difference regarding the focus on theory. Students should leave an engineering education – specifically a computer science education – with an understanding of the methods of development in industry and not only a theoretical understanding.

For reference, the typical Computer Science curriculum looks a little something like this.



UNIVERSITY of VIRGINIA
COMPUTER SCIENCE

Rice Hall
85 Engineers Way
Charlottesville, VA 22904
www.cs.virginia.edu
434.982.2200
cs-office@virginia.edu

Sample BSCS Curriculum Schedule

First Semester - 15 Credits

APMA 1110	Single Variable Calculus (4)
CHEM 1610	Intro Chemistry I for Engineers (3)
CHEM 1611	Intro Chem. I for Engineers Lab (1)
ENGR 1620	Introduction to Engineering (3)
ENGR 1621	Intro. to Engineering Lab (1)
STS 1500 or HSS elective	Science, Tech. & Contemporary Issues or HSS elective1 (3)

Second Semester - 17 Credits

SCI elective	Science elective[2] (3)
HSS elective1	HSS Elective or Science, Tech. & or STS 1500 & Contemporary Issues (3)
APMA 2120	Multivariate Calculus (4)
PHYS 1425	Physics I: Mechanics, Thermo.(3)
PHYS 1429	Physics I Workshop (1)
CS 111x or CS 1120	Introduction to Programming or Introduction to Computing (3)

Third Semester - 16 Credits

APMA course	APMA elective3 or APMA 3100 (3)
HSS elective	HSS elective[1] (3)
CS 2110	Software Develop. Methods (3)
CS 2102	Discrete Mathematics (3)
PHYS 2415	General Physics II: E&M & Lab (3)
PHYS 2419	General Physics II Workshop (1)

Fourth Semester - 16 Credits

STS 2xxx/3xxx	STS 2xxx/3xxx elective (3)
UE elective	Unrestricted elective[4] (3)
CS 2150	Prog. & Data Representation (3)
CS/ECE 2330	Digital Logic Design (3)
CS 2190	CS Seminar[6] (1)
CS 3102	Theory of Computation (3)

Fifth Semester - 18 Credits

APMA course	APMA elect.[3] or APMA 3100 (3)
HSS elective	HSS elective[1] (3)
UE elective	Unrestricted elective[4] (3)
CS elective	CS elective (3)
CS 3330	Computer Architecture (3)
CS 4102	Algorithms (3)

Sixth Semester - 15 Credits

APMA course	APMA elective[3] or APMA 3100 (3)
UE elective	Unrestricted elective[4] (3)
HSS elective	HSS elective[1] (3)
CS elective	CS elective (3)
CS 3240	Advanced Software Develop. (3)

Seventh Semester - 15 Credits

CS elective	CS elective (3)
CS 4970 or CS elective	Capstone Practicum I5 or CS elective (3)
UE elective	Unrestricted elective[4] (3)
CS 4414	Operating Systems (3)
STS 4500	STS and Engineering Practice (3)

Eighth semester – 15 Credits

CS 4971 or 4980	Capstone Pract. II or Capstone Res. (3)
CS elective	CS elective (3)
UE elective	Unrestricted elective[4] (3)
HSS elective	HSS elective[1]
STS 4600	Engineer, Ethics & Prof. Society (3)

Footnotes

1. Chosen from the approved list available in A122 Thornton Hall.
2. Chosen from: BIOL 2010, 2020; CHEM 1620; ECE 2066; MSE 2090; and PHYS 2620.
3. Chosen from APMA 2130, 3080, 3100, 3120 or 3150 (but cannot take both 3120 and 3150).
4. Unrestricted electives may be chosen from any graded course in the University except mathematics courses be- low MATH 1310 and courses that substantially duplicate any others offered for the degree, including PHYS 2010, 2020; CS 1100, 1200; or any introductory programming course. Students in doubt as to what is acceptable to satisfy a degree requirement should get the approval of their advisor and the dean's office, located in A-122 Thornton Hall. APMA 1090 counts as a three-credit unrestricted elective.
5. The CS capstone experience 4970 and 4971 requires 4th year standing.
6. CS 2190 requires second- or third-year standing

Figure 2. Undergraduate Computer Science curriculum for student at University of Virginia (Bachelor).

As seen from the figure, there is only one course in which students may learn industry practices at the University of Virginia – CS 3240. Moreover, this is common for the majority of undergraduate computer science curriculums nationwide. This is simply not ideal for a degree program in which an overwhelming majority of graduates go directly into industry. Although there is an argument for the fact that the university has an obligation to provide a breadth of education as opposed to catering to specifications like a trade school would do, there is more that universities can do to better prepare graduates for industry without wavering on this core mission.

However, this is not all the responsibility of the universities. In order to be accredited and have respected degree programs, universities are forced to meet ABET accreditation standards. This puts universities in a bind even if they do want to increase exposure to industry practices in their curriculums. The current ABET standards for computer science curriculum are as follows:

5. Curriculum

The curriculum requirements specify topics, but do not prescribe specific courses. These requirements are:

Computer science: At least 40 semester credit hours (or equivalent) that must include:

1. Substantial coverage of algorithms and complexity, computer science theory, concepts of programming languages, and software development.
2. Substantial coverage of at least one general-purpose programming language.

3. Exposure to computer architecture and organization, information management, networking and communication, operating systems, and parallel and distributed computing.

4. The study of computing-based systems at varying levels of abstraction.

5. A major project that requires integration and application of knowledge and skills acquired in earlier course work.-

Mathematics: At least 15 semester credit hours (or equivalent) that must include discrete mathematics and must have mathematical rigor at least equivalent to introductory calculus. The additional mathematics might include course work in areas such as calculus, linear algebra, numerical methods, probability, statistics, or number theory.

At least six semester credit hours (or equivalent) in natural science course work intended for science and engineering majors. This course work must develop an understanding of the scientific method and must include laboratory work.

As you can see, there is a heavy emphasis on theoretical coursework, and little to no emphasis on coursework which could prepare students for industry practices. Because of this, this issue cannot be solved on a university-to-university level, it must be solved on a national level with changes to the ABET accreditation standards and ideals on what is an acceptable computer science education.

Bridging the Gap: Changing Engineering Education with A Multilevel Perspective

The multilevel perspective on sustainable transitions presented by Geels is the most useful in analyzing and discovering a solution to the sociotechnical issue of the gap between an engineering education and the engineering workplace. The non-linear transitional models which this model suggests applies to changing an engineering education because this process is one

which is non-linear as well. Furthermore, this model is multifaceted in the way it approaches the problem which is extremely useful. This is a brief description of the model and its core ideas from its paper by Geels:

> The multi-level perspective (MLP) is a middle-range theory that conceptualizes overall dynamic patterns in socio-technical transitions.1 The analytical framework combines concepts from evolutionary economics (trajectories, regimes, niches, speciation, path dependence, routines), science and technology studies (sense making, social networks, innovation as a social process shaped by broader societal contexts), structuration theory and neo-institutional theory (rules and institutions as 'deep structures' on which knowledgeable actors draw in their actions, duality of structure, i.e. structures are both context and outcome of actions, 'rules of the game' that structure actions). These theoretical micro-assumptions have been articulated elsewhere (Geels, 2004; Geels and Schot, 2007, 2010).

This perspective allows for educators to look at the transition as a fluid process and implement changes in a corresponding way. Educators do not need to change the national curriculum standards at once, however, courses can be added and changes can be made over time in an iterative way. This will also give educators, students, and employers the opportunity to adapt over time as they will not have to adjust to an entirely new curriculum standard at once. Furthermore, viewing the transition with respect to the three analytical levels of the Multilevel perspective allows for educators to make stable changes. Educators can focus on the more stable levels first and continue to make more changes and advance the transition as these stable changes are implemented and accepted as the new standard.

However, this multi-level perspective is of no use if educators do not know what should be changed. There are a number of things which contribute to a successful software engineering culture, and these are the aspects which should be added to the Computer Science curriculum so that it better prepares students.

One of the key foundations of a healthy software engineering culture is teamwork – simply because of the way in which scalable software is produced. Teamwork is paramount because a majority of large-scale software products are the result of divided labor between teams in which team members specialize in certain areas. This is widely recognized in the industry and is one of the reasons why the agile methodology is so prominent. Teamwork is one of the key elements of the Agile methodology. It specifically encourages elements of teamwork such as transparency with the team – "There should be transparency of work so that each member is responsible for a portion of the work and know the responsibilities of their team members." (What is agile methodology?)

This is also one of the key aspects which needs to be changed in the computer science curriculum. There is a lack of courses in which students in which a work place is simulated and students can experience how software is produced in the professional industry. Moreover, there is a lack of emphasis on software quality in the computer science curriculum. In industry, quality is paramount – a company cannot deliver a faulty or buggy product under any circumstances. Karl Wiegers, a professional consultant, states this as well when he states "Quality is the top priority; long-term productivity is a natural consequence of high quality." (Wiegers) This must be addressed in the Computer Science education. Students go through the majority of their engineering education without a focus on quality but rather a focus on completion, specifically, completion before a deadline. Sylvia Stuurman, a computer science professor in the Netherlands,

discusses this in her research paper – Research plan: Software Quality in Education. She specifically discusses how it is the result of a lack of enthusiasm.

> Lack of enthusiasm for software quality On the one hand, students find it difficult, or perhaps boring, to pay attention to software quality. Testing their code is tedious; students work towards a result and are happy when their program seems to 'work'. It seems superfluous to pay much attention to a good design, and to maintainable code. Finding ways to stimulate students' enthusiasm for software quality is one of the goals of our research. (Stuurman, 2)

Although deadlines do exist in the real world, there are often expectations which are not as concrete as those which students experience in the academic environment. Courses which address this gap should be added to the curriculum. For example, courses in which students develop a semester long project which could be published in some manner should be encouraged and emphasized as a part of Computer Science curriculums at a larger scale. One such course offered at the University of Virginia is CS 4720 – Mobile App Development, and from personal experience, this was one of the most practical classes in the undergraduate computer science curriculum.

Wiegers also states that continuous education and measuring productivity are key foundations of a healthy software engineering culture (Wiegers). These are also foundations which are not found in the traditional Computer Science curriculum. These foundations require a level of self-sufficiency and independence which does not get developed with the curriculums currently in place. Courses which encourage self-sufficiency and independence in this fashion should be added to the curriculum. For example, classes in which students are given a large

project to complete but little guidance could not only better reflect industry practices but encourage the removal of handholding at an early stage to ensure that students are ready to contribute independently when they arrive in industry.

Now that the gap has been analyzed and what needs to be changed has been cemented, the question of what the best way to bridge the gap is must be addressed.

A Radical Proposal for Change: A Trade School Approach

As stated earlier, a computer science education which aligns more closes with industry expectations for software engineers would better achieve the goal of an engineering degree – enabling students to become successful engineers who will be technical problem solvers. Some research even suggested very drastic measures to achieve this such encourage computer science to be a trade profession or allowing the university to offer a degree specifically in software engineering. Specifically, the Wall Street Journal believes computer programming is a trade profession – "Computer programming, in other words, has become a trade. Like nursing or welding, it's something in which a person can develop at least a basic proficiency within weeks or months. And once budding coders learn enough to get their first jobs, they get onto the same path to upward mobility offered to their in-demand, highly paid peers." (Press)

Although this is a very unconventional suggestion for a university, there are some benefits which would warrant its consideration. Namely, employability. Trade school professions have always been held in high regard with respect to employability due to the fact that graduates exit the school with useful, immediately applicable skills. This has not always been the case with college degrees.

However, this radical suggestion is not possible in its entirety, so more feasible or partial solutions must be analyzed. This research suggests that some aspects of a typical trade school

education be implemented into the engineering degree programs for computer science, and that they be implemented in the manner described in the Outline for Changes section.

This deliverable is an outline of how the computer science curriculum can be changed to better map to industry standards and practices and allow for engineering graduates with computer science to be better prepared to enter the workforce and offer an immediate contribution. These are the changes which should be made to engineering curriculums nationwide.

Outline for Changes

1. Add more specified coursework
    a. Examples for Computer Science Degree
        i. Agile Methodology
        ii. Proof Of Concept to Production (Pipeline Course)
        iii. Producing Software at Scale – An exercise in team-based development
2. Add more hands-on coursework with less professor involvement. Allow students to learn on their own.
3. Introduce coursework which focuses on the development of soft skills (Garousi 6).
4. Add more project-based courses as this is how work is commonly done in industry. Little to no work is truly done individually (Garousi 6) (Conde 6).
5. Add coursework in which students can learn from people in industry and not just from professors.

6. Allow students to have more freedom in their curriculum and the coursework which they take. Make curricula less rigid.
7. Offer more specifications in the curriculum and tracks for students.
    a. Examples for Computer Science
        i. Cybersecurity
        ii. Web Development
        iii. Information Systems
        iv. Artificial Intelligence

Conclusion

This research digs into the question of why the gap between engineering education and engineering practice has become so large, and what measures can be taken to address it. It goes into the specifics of this issue as it relates to the traditional computer science curriculum, its mapping to the software engineering industry, and how it can be solved at universities.

Specifically, this research generated a new understanding of how computer science can be taught in university settings which would allow it to better track into industry while still allowing for universities to achieve their fundamental mission of providing students with a deep, theoretical knowledge of the subject. The new understanding was generated based upon prior research done on the computer science curriculum and software engineering industry and a unique look into how trade schools operate with the fundamental intent of providing graduates with opportunities for immediate employment. This research looked at what elements of a trade school education could be translated into the university environment successfully without comprising the mission of universities.

This new understanding was the basis for the deliverable of this research – the outline for change – discussed in the previous section. The outline provides numerous clear, action items which universities could implement that would significantly lessen the gap between what knowledge computer science students graduate with in comparison to what they are expected to know entering the software engineering industry. The outline also only suggests changes which would not change the parts of the curriculum which achieve the university's core mission of providing students with deep, theoretical knowledge. It simply suggests the incorporation of some aspects of a trade school education along with other suggestions – based on research into the computer science curriculum and software engineering industry – that would allow the curriculum to better prepare students for industry.

Although this research had a specific focus on computer science, much of the research done and proposals suggested can be translated to other fields in engineering in which there are similar disparities between industry and education. Using this research as a template, researchers can look to close the gap between engineering education and engineering practice on a grander scale and allow for engineers to graduate and better achieve their goals of producing meaningful change using their technical abilities.

References

Akter, S., McCarthy, G., Sajib, S., Michael, K., Dwivedi, Y. K., D'Ambra, J., & Shen, K. N. (2021). Algorithmic bias in data-driven innovation in the age of ai. International Journal of Information Management, 60, 102387. https://doi.org/10.1016/j.ijinfomgt.2021.102387

Bachelor of Science in Computer Science (BSCS ... https://engineering.virginia.edu/sites/default/files/common/departments/computer-science/files/2-%20BSCS%20Info%20and%20Curriculum%202017%20ws.pdf.

Boer, R. de. (2019, September 17). Understanding the role of AI bias in Healthcare. Artificial Intelligence in Healthcare & Radiology. Retrieved November 1, 2021, from https://www.quantib.com/blog/understanding-the-role-of-ai-bias-in-healthcare.

Brendel, A. B., Mirbabaie, M., Lembcke, T.-B., & Hofeditz, L. (2021). Ethical Management of Artificial Intelligence. Sustainability, 13(4), 1974. https://doi.org/10.3390/su13041974

"Career and Salary of Software Engineer." Code a New Career | ComputerScience.org, 21 Mar. 2022, https://www.computerscience.org/software-engineering/careers/software-engineer/career-and-salary-outlook/.

Conde, Javier, et al. "Bridging the Gap between Academia and Industry through Students' Contributions to the FIWARE European Open-Source Initiative: A Pilot Study." Electronics, vol. 10, no. 13, June 2021, p. 1523. Crossref, https://doi.org/10.3390/electronics10131523.

Chao, C. (2019). Ethics Issues in Artificial Intelligence. 2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI), 1-6.

Garousi, Vahid, et al. “Closing the Gap between Software Engineering Education and Industrial Needs.” IEEE Software, vol. 37, no. 2, 2020, pp. 68–77., https://doi.org/10.1109/ms.2018.2880823.

Geels, F.W. 2011. The multi-level perspective on sustainability transitions: Responses to seven criticisms. Environmental Innovation and Societal Transitions. 1, 1 (2011), 24–40.

Home: https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2021-2022/. Accessed: 2022-03-23.

Johnson, G.M. (2020). Algorithmic bias: on the implicit biases of social technology. Synthese, 198, 9941-9961.

Kose, U., & Vasant, P.M. (2017). Fading intelligence theory: A theory on keeping artificial intelligence safety for the future. 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), 1-5.

Köse, U. (2018). Are We Safe Enough in the Future of Artificial Intelligence? A Discussion on Machine Ethics and Artificial Intelligence Safety. BRAIN. Broad Research In Artificial Intelligence And Neuroscience, 9(2), pp. 184-197.

Li, X., & Zhang, T. (2017). An exploration on artificial intelligence application: From security, privacy and ethic perspective. 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), 416-420.

May, Elizabeth, and David S. Strong. “Is Engineering Education Delivering What Industry Requires.” Proceedings of the Canadian Engineering Education Association (CEEA), 2011, https://doi.org/10.24908/pceea.v0i0.3849.

“Our Mission, Vision and Core Values.” University of Virginia School of Engineering and Applied Science, 9 Nov. 2021, https://engineering.virginia.edu/about/mission-

vision#:~:text=Our%20mission%20is%20to%20make,leaders%20to%20solve%20global%20challenges.

Ostrowski, D. (2018). Artificial Intelligence with Big Data. 2018 First International Conference on Artificial Intelligence for Industries (AI4I), 125-126.

Press, ILLUSTRATION: Associated. “Computer Programming Is a Trade; Let's Act like It.” The Wall Street Journal, Dow Jones & Company, 4 Aug. 2014, https://www.wsj.com/articles/computer-programming-is-a-trade-lets-act-like-it-1407109947.

Rogozea, L. (2009). Towards ethical aspects on artificial intelligence. Smith, J. (2019). COMPUTATIONAL THINKING WITHOUT ALGORITHMIC BIAS. ICERI2019 Proceedings.

Schilling, Jan, and Ralf Klamma. “The Difficult Bridge between University and Industry: A Case Study in Computer Science Teaching.” Assessment & Evaluation in Higher Education, vol. 35, no. 4, 2010, pp. 367–380., https://doi.org/10.1080/02602930902795893.

Stuurman, Sylvia, et al. “Research Plan: Software Quality in Education.” https://research.sylviastuurman.nl/.

Sun, W., Nasraoui, O., & Shafto, P. (2020). Evolution and impact of bias in human and machine learning algorithm interaction. PLOS ONE, 15(8). https://doi.org/10.1371/journal.pone.0235502

Wiegers, Karl. “Building a Healthy Software Engineering Culture.” Medium, The Startup, 15 Sept. 2021, https://medium.com/swlh/building-a-healthy-software-engineering-culture-59183b93389d.

"What Is Agile Methodology? Benefits of Using Agile." Nvisia, https://www.nvisia.com/insights/agile-methodology.

The Case for Ethical Artificial intelligence with Minimal Bias

Closing the Gap between Engineering Education and Engineering Practice

A Thesis Prospectus
In STS 4500
Presented to
The Faculty of the
School of Engineering and Applied Science
University of Virginia
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By
Rohit Batra

November 1, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISORS

Kathryn A. Neeley, Department of Engineering and Society

Daniel Graham, Department of Computer Science

The issues with Artificial Intelligence and their sources:

Bias is defined as a prejudice or tendency to believe towards or against something in a way which is often unfair. With the vast amounts of different opinions and social cultures present in today's world, it is the case that bias is unavoidable and something which must be dealt with, and biases have been handled over time as our cultures continue to develop and advance socially. However, with the advancement of technology and development of intelligence systems – systems designed to mimic human intelligence – it has also become inevitable that biases will develop in those intelligent systems. These biases present in the intelligent systems have been coined algorithmic biases, and they contribute and lead to the creation of unethical systems of artificial intelligence.

Though, these biases are a major problem going forward in the development of artificial intelligence and machine learning systems, there are also other problems which have mistakenly been grouped together into this issue of bias. Some of these issues are a lack of privacy and security regarding user data and a misunderstanding of what these systems do from the point of view of the general public.

However, in contrast to the social systems humans have in place to keep these biases in check, these intelligent systems being developed currently have no systems in place to regulate the biases which they may form, and this could present a dangerous threat if it continues to progress without regulations and checks in place. With the rise of consumer technology and collection of consumer data, along with the advancement of artificial intelligence, this presents an especially dire concern which must be addressed. A failure to take action and bring attention to this issue will result in dangerous biases developing in systems of artificial intelligence, and they will fail to serve their desired purpose of aiding society. A portion of this paper will discuss

potential solutions to this issue involving implementing standardized, methodical ways to develop artificial intelligence and collect user data in a way such that limits biases from forming in the first place, and also discussing ways to eliminate the biases already present in the large scale artificial intelligence systems in place today.

Another portion of this paper will discuss the main source of these issues: the growing disparity between an engineering education and engineering practice. In this paper, Engineering Practice will be defined as the ongoing and fruitful pursuit of technical knowledge for the betterment of society as a whole. The technical portion of this paper will discuss this definition of engineering practice, and it will also dive into a more specific subset of engineering practice and engineering education which is software engineering and computer science and how they differ in ways which they should not.

Looking at Bias, Security, and Privacy in Relation to the Growing Use of Consumer Data for Artificial Intelligence and Machine Learning:

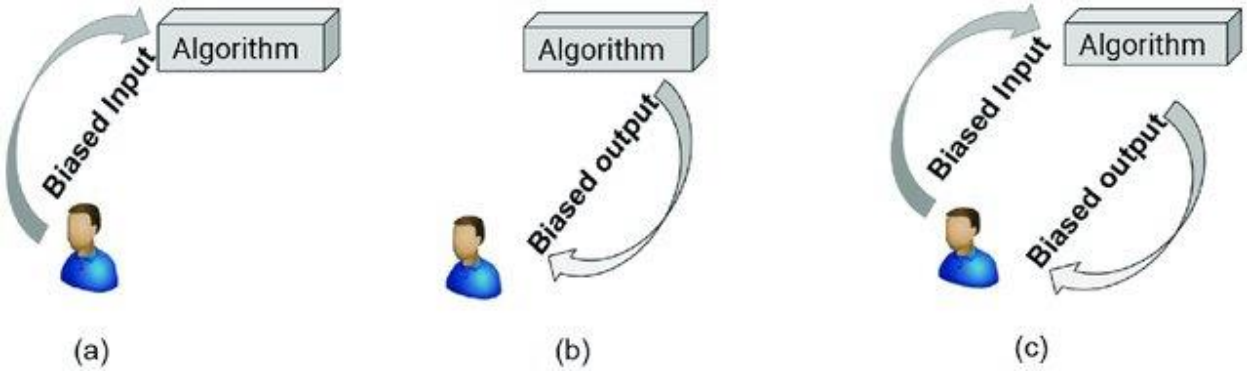
Capitalistic companies have one primary motive: to make a profit. Generally, in well-regulated industries, these companies can be kept in check and be forced to act ethically despite it not always being in the best interest of trying to make a profit. However, with the rapid pace of innovation in the technology industry, regulation has failed to keep up, and this has allowed for the key players in the industry to pursue profit in ways which do not keep the consumers' safety in mind. The key companies in the industry such as Google and Facebook have had full reign over what they can produce. Although a lack of pace in terms of regulation keeping up has been an issue in other industries as well, the destabilizing condition when it comes to artificial intelligence and machine learning which makes this issue so dire is the growing use of these

products in consumer settings and the massive potential for harm which they have in such settings.

Specifically, a key socio-technical issue which has resulted from this lack of regulation is a conflict between users and companies when it comes to the collection of their data (Brendel 5). Most consumers are unaware of the fact that their data is being collected in mass and being used to develop artificial intelligence systems which may operate in unethical ways. This has resulted in an immense distrust between most consumers and the large technology companies who are creating these artificial intelligence systems. Consumers have no idea what their data is being used for, how their privacy is being invaded, or even where their data is going and the security of it.

Moreover, due to the lack of regulation, companies have no incentive to filter or even check the user data which they are collecting. This has led to a system in which the companies collected biased data and use the biased data to develop systems of artificial intelligence which leads to AIs with unethical biases. It is fairly evident that this is a cyclic system, and without intervention, algorithmic biases will only continue to worsen.

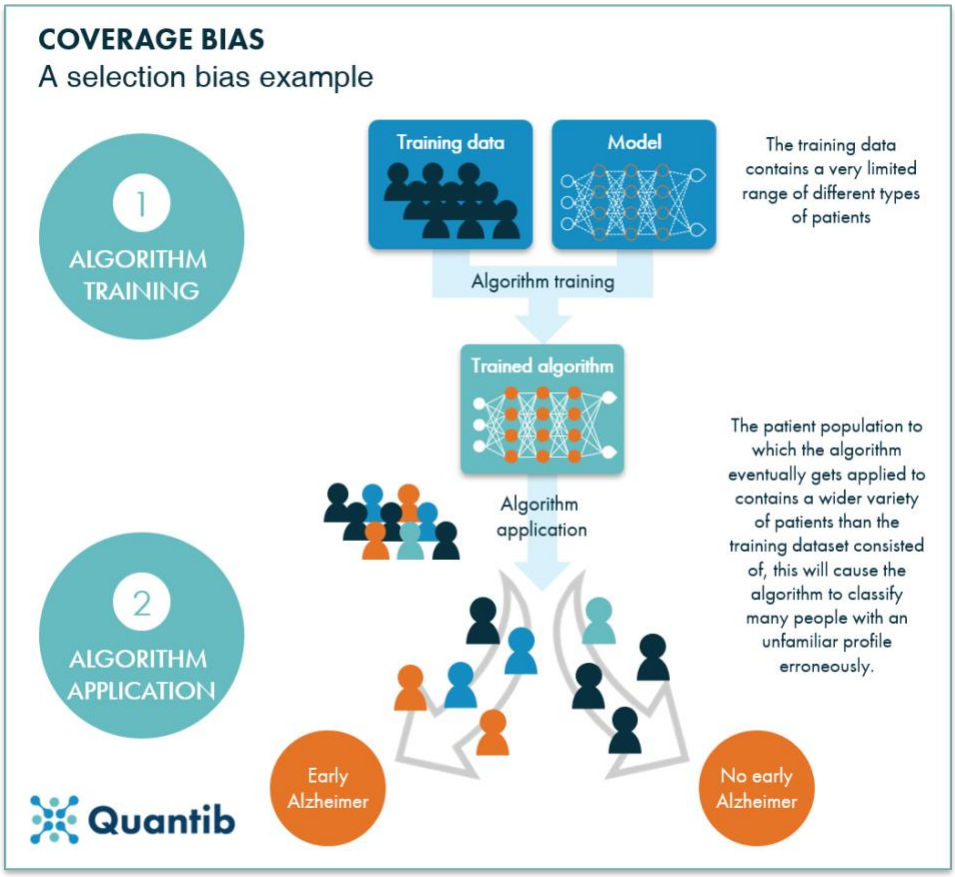
Source & Cyclic Nature of Algorithmic Bias



This figure provides a clear illustration showing the flaw in the current system (Sun).

Although these issues do not pose an imminent threat as of right now, there is much evidence to showcase that if companies continue to operate and develop AIs in such respects, they have the potential to be very dangerous. As they continue to advance and do more and become more intelligent, the inherent biases which they have will only get more dangerous as there will be more avenues for them to be acted upon (Brendel 1). At the current moment, AI cannot and does not do all that much, but many of these companies have wide-eyed aspirations for the AIs which they are working on, so it is paramount that a systematic way to develop AIs without biases and AIs which can reason about any biases which they do develop is put into place.

Coverage Bias and its' affects in the real world



This figure provides a more specific example and shows a specific consequence of algorithm bias in the healthcare industry (Boer).

My research will attempt to resolve these issues from a sociotechnical standpoint by outlining a way in which trust can be established between companies and consumers regarding the collection of their data and a way in which data can be collected and filtered such that it prevents biased AIs from developing in the first place. My research will also look at a comparison of solutions involving minimizing bias, solutions involving eliminating bias, and solutions involving compensating for bias and try to determine which is the most effective. Furthermore, one of the key causes of this issue in the first place is a lack of ethical and sufficiently educated engineers. The deliverable of my STS research will be a comparison of the different solutions and their efficiencies and efficacies regarding achieving ethical artificial intelligence and machine learning. Moreover, if the average engineer at one of the companies developing these systems was able to reason about the negative outcomes more thoughtfully, they would be able to prevent many of these issues in the first place. However, due to the disparity between an engineering education and what is required in the workplace, this check has failed in recent times. The technical portion of my paper will address this further.

The Disparity Between Computer Science (Engineering Education) and Software Engineering (Engineering Practice) – Bridging the Gap:

Over the years as the engineering industry has experience rapid growth, the disparity between what engineers need to know for the job and what engineers come out of school knowing has becoming increasingly apparent. This portion of the paper will discuss this issue in specific as it relates to the disparity between a traditional Computer Science education and what the Software Engineering industry demands of Computer Science graduates who enter industry.

The destabilizing condition which makes this issue so important at this time is increasing use of commercial software along with the lack of sufficiently educated engineers resulting in

inefficient, insecure, and insufficient software products. This can be seen very clearly in the case described earlier of unethical artificial intelligence and machine learning systems. A sufficient supply of ethical, talented, and properly-educated engineers who methodically develop software could have prevented many of the issues described.

However, that is only one case in which the disparity is evident, and it is not the root cause of the issue. In my experience, there are a few key differences which I believe, if corrected, would be beneficial to the software industry as a whole. Though, all of these differences will involve a reevaluation of what the purpose of an engineering education is and how it differs from the purpose of a traditional liberal arts education.

One of the main differences which I experienced personally with my internship experience is the focus on theory in the educational environment and the lack of adequate exposure to applicable workplace practices. There are a plethora of courses available on theoretical concepts, however, if a student wants to learn about relevant and modern day software engineering practices, there are only a handful of courses available. I believe this needs to change in the engineering curriculum. Students should be able to enter the industry with a sufficient understanding of what will be required of them to be a valuable employee for the company from day 1. I experienced this personally in my internship in the software engineering industry when I was given no hand-holding with many of the tools which are used in the industry-standard today such as Git – I had to learn all of these things on the fly as I was completing my project which I thought was very difficult. I believe that if this change can be implemented into many of the engineering curriculums, it will benefit engineers who are looking to enter industry as they will be more well-equipped for what will be required of them.

Another key difference between a typical computer science curriculum and software engineering is the speed and way in which projects are completed. In the academic environment, projects or assignments typically have one strict deadline and must be submitted in full at that deadline. This differs from the common practice in industry where things are done in an Agile way. The Agile methodology is a practice which was put into place in the software engineering industry in which software is developed in a periodical and cyclic way. In my personal experience, it was difficult to adapt to this method coming from the academic environment where I was able to commonly crank out assignments rapidly when the due date was approaching. With the Agile method, I had to learn how to build software in a feature by feature manner in a way such that it could be constantly and consistently iterated on.

Both of these differences are things which should be addressed in the Computer Science curriculum to sufficiently prepare graduates to enter industry, however, they also showcase the larger problem as a whole – how an engineering education does not sufficiently prepare graduates for industry. I will attempt to address this problem with my technical research.

My technical research will involve researching the origins of engineering education, reevaluating its applicability in today's world, and coming up with a deterministic way in which engineering educations can be improved worldwide to better prepare graduates who are looking to enter industry. My deliverable will be an outline of key changes which can be implemented to engineering curriculums at a large scale that will allow for graduates to be better prepared to enter industry.

Looking at Solutions:

In an attempt to solve these issues, my research will have two key deliverables. The anticipated deliverable of my technical work will be an outline which contains key changes

which can be implemented to standard engineering educations which are in place today that will allow for engineers to be better suited to enter the workforce and industry. The anticipated deliverable of my sociotechnical research will be an outline of how trust can be established between consumers and companies regarding data collection, and also a comparison of different solutions regarding the removal of bias from artificial intelligence and machine learning. These solutions will allow engineers to come away with a better understanding of the societal impact of the work which they are doing – both positive and negative aspects – and also a better understanding of what they can do from a technical aspect to prevent the negative societal impacts of their work. The deliverables will allow for educational professionals to look at some changes which they could make to engineering curriculums which would benefit engineering practice as a whole, and they will also allow for engineers to learn a better way of going about the development of AIs and collection of user data which will in turn prevent unethical and biased AIs from coming about.

References

Akter, S., McCarthy, G., Sajib, S., Michael, K., Dwivedi, Y. K., D'Ambra, J., & Shen, K. N. (2021). Algorithmic bias in data-driven innovation in the age of ai. International Journal of Information Management, 60, 102387. https://doi.org/10.1016/j.ijinfomgt.2021.102387

Boer, R. de. (2019, September 17). *Understanding the role of AI bias in Healthcare*. Artificial Intelligence in Healthcare & Radiology. Retrieved November 1, 2021, from https://www.quantib.com/blog/understanding-the-role-of-ai-bias-in-healthcare.

Brendel, A. B., Mirbabaie, M., Lembcke, T.-B., & Hofeditz, L. (2021). Ethical Management of Artificial Intelligence. *Sustainability*, *13*(4), 1974. https://doi.org/10.3390/su13041974

Chao, C. (2019). Ethics Issues in Artificial Intelligence. 2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI), 1-6.

Johnson, G.M. (2020). Algorithmic bias: on the implicit biases of social technology. Synthese, 198, 9941-9961.

Kose, U., & Vasant, P.M. (2017). Fading intelligence theory: A theory on keeping artificial intelligence safety for the future. 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), 1-5.

Köse, U. (2018). Are We Safe Enough in the Future of Artificial Intelligence? A Discussion on Machine Ethics and Artificial Intelligence Safety. BRAIN. Broad Research In Artificial Intelligence And Neuroscience, 9(2), pp. 184-197.

Li, X., & Zhang, T. (2017). An exploration on artificial intelligence application: From security,

privacy and ethic perspective. 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), 416-420.

Ostrowski, D. (2018). Artificial Intelligence with Big Data. 2018 First International Conference on Artificial Intelligence for Industries (AI4I), 125-126.

Rogozea, L. (2009). Towards ethical aspects on artificial intelligence.

Smith, J. (2019). COMPUTATIONAL THINKING WITHOUT ALGORITHMIC BIAS. ICERI2019 Proceedings.

Sun, W., Nasraoui, O., & Shafto, P. (2020). Evolution and impact of bias in human and machine learning algorithm interaction. *PLOS ONE*, *15*(8). https://doi.org/10.1371/journal.pone.0235502