Holistic QA: Software Quality Assurance for the Machine Learning Era

Shane Downing and M. Affan Badar
Department of Applied Engineering and Technology Management
Indiana State University
Terre Haute, IN 47809, USA
sdowning8@sycamores.indstate.edu, M.Affan.Badar.indstate.edu

Abstract

The traditional software space (1.0) has seen more than fifty years of creation, testing, and delivery of deterministic software, but this tradition is being disrupted by machine learning (2.0). However, ML and traditional software are considerably different, and the nascent ML industry uses unique workflows and toolsets for both life cycles. This "one foot in each raft" scenario forces companies to support duplicate resources which are essentially doing the same thing. This paper proposes an answer to the research question: Can software 2.0 quality assurance be performed effectively using the same process as 1.0? A systematic literature review was performed and process documents from a machine learning company reviewed. 132 papers were selected and refined to 24. ML process documents supported what is an industry standard ML life cycle. While the literature review shows a gap relative to holistic QA solutions for ML products, process documents showed the answer to the research question is...perhaps, since it was determined that the typical ML life cycle can be mapped to the standard software 1.0 life cycle. This paper proposes using that mapping to extend an existing 1.0 QA process architecture, the Quality Assurance Machine (QAM), to effectively manage both 1.0 and 2.0 QA.

Keywords

Quality assurance, traditional software, machine learning, systematic literature review

1. Introduction

The software industry is in transition: The traditional space, software 1.0, has seen more than fifty years of creation, testing, and delivery of deterministic software, but this tradition is being disrupted by the stochastic model of software 2.0. Data driven software development, a paradigm shift, feeds massive amounts of cleaned and curated data to models which create software to solve previously unsolvable problems. The computer is no longer told what and how to do something, it is told what the answer should be, then given data and a model to create software to arrive at it. These ideas are not new, they are decades old. What is new, and what has turned ideas into reality, is the introduction of big data and more than a doubling of compute power. Digitally connected systems produce 2.5 quintillion (i.e., 2.5×1018) bytes of data per day, and some is used by software 2.0. Compute power comes courtesy of the gaming industry: Graphics Processing Units (GPUs) developed to enable graphics and video rendering was the power needed to process big data and unleash the mystery of machine learning (ML). Mystery, because the algorithms created by ML are complicated and difficult for humans to understand, and details of how answers are created are not well known. ML solutions tend to be a complex black box, and this has created issues for companies developing, and anxiety for those using, the solutions. Software 1.0 has a rich set of processes and tools for development and quality assurance which have grown over decades. However, 2.0 is very new, and a problem faced by the industry is 2.0 quality assurance processes and tools, those which give companies and consumers confidence, are lagging.

Within industry, machine learning model development follows a pattern: acquire, clean, and curate data, train and test a model, deploy the model. This process, while not backed by industry standard tools and processes, is common. However, ML models do not typically exist in isolation. A current deployment method for software 2.0 is to wrap ML models with software 1.0 infrastructure, thus creating reliance on both technologies simultaneously. But ML and traditional software are considerably different and require unique workflows and toolsets. This "one foot in each raft" scenario forces companies to support resources which, at a high level, are doing the same thing: creating and delivering software. This paper answers the research question: Can quality assurance be performed effectively for ML solutions

using the same process as that for software 1.0? An investigation is performed into the feasibility of extending an existing quality assurance architecture, the Quality Assurance Machine, to support software 1.0 and 2.0 simultaneously.

2. The Quality Assurance Machine (QAM)

Quality is a simple concept and can be defined by a single sentence: Quality is the degree to which a system, component, or process meets specifications and user requirements (Defeo, 2016). This definition of quality is generic, applying to everything from apples to Zoom software. But while the definition of quality is simple, the planning, inspection, and enforcement of quality can be a fuzzy, abstract space. As seen from the definition, half of the target...meeting user requirements, can be subjective. This subjectiveness transforms what should be an objective assessment into a muddled space with unexpected factors, such as packed phrases and politics. Quality is simple in the abstract, but confounding in practice, and software quality is representative of this confounding space.

Since the 1960s, software development and its associated quality activities have followed the general process flow of Requirements, Design, Develop, Test, and Release (RDDTR) (Kneuper, 2009). There have been modifications to the internal workings of this flow, waterfall to spiral to agile, but software creation continues to follow the RDDTR paradigm. This is not surprising, as the goals of delivering software are to meet specifications and user requirements, and the generic phases of RDDTR are focused on both. Empirically speaking, software quality innovation tends to lag software creation innovation, advances in quality being more of a response to evolving software development methodologies than novel approaches. But what if quality's focus was not one of catching up to the shifting development landscape, but instead was on something that has not really changed? What if the focus was to ensure process quality and software quality in each of the RDDTR phases? Would there be value in a quality assurance architecture that:

1) Simultaneously addresses two fundamental tenets of software quality: Quality Assurance (QA) and Quality Control (QC).
2) Creates logical connections between the major components of the software quality process.
3) Presents software quality assurance in the vernacular.

Empirical evidence suggests the answer to these questions is yes, and the architecture which accomplishes this is the Quality Assurance Machine (QAM).

Software Quality Assurance (SQA) is defined as…
A set of activities that define and assess the adequacy of software processes to provide evidence that establishes confidence that the software processes are appropriate for and produce software products of suitable quality for their intended purposes. A key attribute of SQA is the objectivity of the SQA function with respect to the project. The SQA function may also be organizationally independent of the project; that is, free from technical, managerial, and financial pressures from the project (IEEE, 2014).

Decomposing, Software Quality Assurance contains forward-looking *process* (QA) and retrospective *product* (QC) concerns. QA is a set of activities that use audits, metrics, and tools to ensure software life cycle processes work to prevent defects. Process requirements are, among other things, risk mitigation efforts - a forward-looking process view of quality. SQA also determines if software products are of suitable quality for their intended purposes. Are there defects in the software that's already been constructed? This retrospective product view of quality is Quality Control. The Quality Assurance Machine supports both QA and QC quality views.

The QAM was designed to address persistent quality issues seen across multiple companies developing software from the lowest level (firmware) to the highest (cloud applications). Using a reference model of a machine with engines encapsulating specific SQA activities has multiple benefits, with perhaps the most prominent being strategic. The image of a machine with engines provides a focal point for the quality role within an organization and moves the SQA focus from a team to a functional perspective, thus putting a product's quality function on the same visible level as the product itself, as shown in Figure 1.

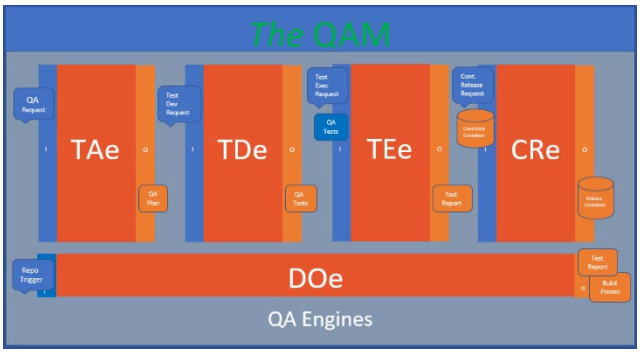

Figure 1. The Quality Assurance Machine (QAM)

QAM Engines
TAe (Test Analysis engine) creates QA plans
TDe (Test Development engine) develops test coverage to support QA plans
TEe (Test Execution engine) executes QA plan tests and logs defects
CRe (Container Release engine) inspects and releases product containers and archives
DOe (DevOps engine) creates QAM automation

2.1 Empirical Benefits of the QAM

2.1.1 Captures State of Quality

The input to each engine is prepared using audits, metrics, and tools, and this input must conform to process requirements. Therefore, the input to QAM engines is a view of the future of software quality - Quality Future. Engine output is a measure of a software's conformance to requirements, and since requirements are built into the software, it is a view of software quality's past; the output is Quality Past. And since the QAM is a machine, its state can be captured, thus measuring the total state of a software's present quality, or Quality Present. Like A Christmas Carol, a software's quality is composed of Quality Past, Quality Present, and Quality Future, and the QAM captures each.

2.1.2 Reduces Process Variability

All engines follow a common template, and this reduces process variability: inputs to an engine must conform to entrance criteria, which are process specifications. This conformance to specifications is quality control, a view of Quality Future. Automating the quality process, using the DOe, creates efficiencies and reduces variability. Schroeder notes that the well-known quality methodology, Six Sigma, is "an organized, parallel-meso structure used to reduce variation in organizational processes" (Schroeder, 2008). Variability, as noted by Six Sigma advocates, is the root of all process evil.

2.1.3 Reduces Subjectivity

While the quality function can become subjective, each engine aligns with typical software quality assurance expectations:
TAe: Organizations tend to agree changes in a software product should be understood
TDe: Organizations tend to agree software should be tested before release (test development)
TEe: Organizations tend to agree software should be tested before release (test execution)
CRe: Organizations tend to agree software should follow a rigorous release process

Although organizations tend to agree on these quality tenets, implementation and adherence can be shallow. Engine entrance and exit criteria moves the focus from individuals stating their quality case to an inanimate object verifying organizationally agreed-upon process expectations.

2.1.4 Simplifies the Meaning of Quality

The QAM maps quality terminology into the vernacular. QA discussed as a machine with engines that perform work reduces the need to first understand the language of the quality discipline, thus enabling more effective organizational communication.

2.1.5 Integrates with Typical Software Development Methodologies

Since the QAM was designed to support RDDTR, the QAM works with waterfall, V-model, agile, etc.

3. Methodology

The research question as stated in the introduction section is: Can quality assurance be performed effectively for ML solutions using the same process as that for software 1.0? To answer this question, a systematic literature review was performed and process documents from a machine learning software company were reviewed. Systematic literature review has been used by many researchers. For example, Al-Odeh at el. (2021) conducted a systematic literature review spanning 2000-2020 related to sustainable supply chain management. This resulted in 39 articles. Guraja et al. (2022) discovered 17 articles in a systematic review of articles published between 2000 and 2021 which studied the impacts of state budget cuts on public higher education institutions in the US. The objective of this paper's literature review was to determine if similar research exists and where the current focus is for ML quality assurance. The process document review was to understand industry ML software life cycles. Using pre-defined selection criteria and citation network analysis for the literature review, 132 papers related to quality assurance for artificial intelligence software were selected. The complete research methodology is illustrated in Figure 2. Different components of the methodology are discussed in the following sub-sections.

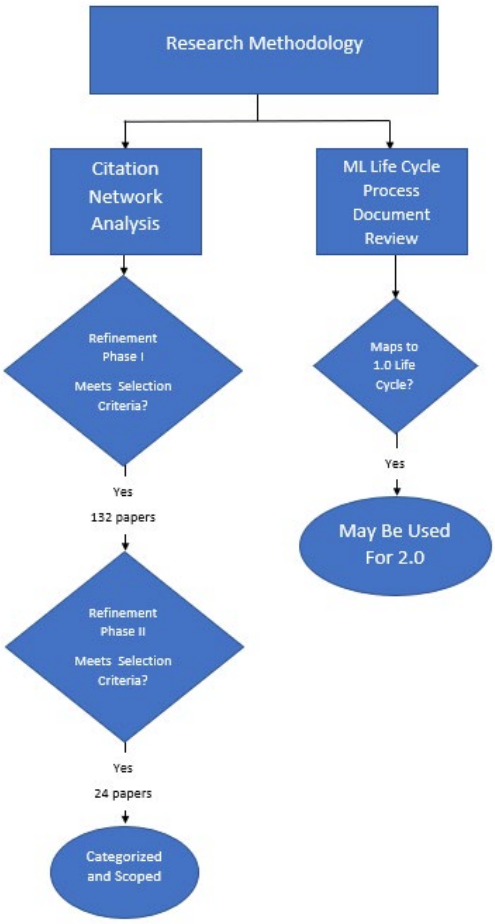

Figure 2. Research Methodology

3.1 Literature Review: Phase I - Citation Network Analysis

According to Khan et al. (2003), step two of the five steps to conducting a systematic review is identifying relevant work. The search for studies should be extensive. Multiple resources (both computerized and printed) should be searched without language restrictions. The study selection criteria should flow directly from the review questions and be specified a priori (Khan et al., 2003). The review conducted for this paper did not use traditional search methods, but rather citation network analysis using Inciteful's paper discovery tool. A traditional review would, as Khan et al. note, search for studies based on a priori criteria that flow directly from the research questions. For this paper, the literature review was performed to determine if holistic ML product QA was being researched. Rather than search multiple databases and scrub hundreds of papers for evidence (unlikely thousands as this field is so new), four seed papers were selected and the search was delegated to, fittingly, algorithms.

Using a seed pool, Inciteful's discovery tool builds a network of papers from citations using network analysis algorithms, an approach used commonly in social network solutions (Inciteful, 2022). The seed pool grows as discovered papers matching pre-selected criteria are added to the pool and a new network created. This process of selecting papers from discovery and growing the network continues until new papers matching the selection criteria are no longer discovered. According to Inciteful, citation analysis presents not only the most important papers in the graph, but also the most similar (Inciteful, 2022).

The seed papers for initial discovery were Quality Assurance for AI-Based Systems (Felderer and Ramler, 2021), Quality Assurance Challenges for Machine Learning Software (Alamin and Uddin, 2021), A Systematic Mapping Study on Testing of Machine Learning Programs (Sherin et al., 2019), and Towards Guidelines for Assessing Qualities of Machine Learning Systems (Siebert et al., 2020). These papers were selected simply for their QA for ML focus. The discovery tool returned many papers with each addition of new papers to the network, but only those matching the following criteria were added:

- Model quality
- Case studies
- Lifecycle
- Mapping concepts 1.0 to 2.0
- Testing techniques
- Quality assurance
- Frameworks
- Literature review
- Process model
- Software 1.0 techniques extended to software 2.0
- ML best practices

As noted, when new papers were no longer discovered, the process ended. This occurred at 132 papers.

3.2 Literature Review: Phase II – Categories

The population of papers from the citation network analysis phase were further refined by reading paper abstracts and again applying the paper selection criteria. This yielded 24 final papers that were separated into the categories as presented in Table 1.

Table 1. Final Papers in Categories

Modifying 1.0 testing for ML	Bosch et al. (2018), Damyanova (2020), Lwakatare et al. (2020), Revanna (2019)
2.0 best practices/lifecycle improvement	Ashmore et al. (2021), Haakman et al. (2021), Serban et al. (2020)
2.0 QA	Akkiraju et al. (2020), Jöckel et al. (2021), Lenarduzzi et al. (2021), Mannarswamy et al. (2020), Poth et al. (2020), Santhanam (2020), Studer et al. (2021)
2.0 challenges	Ishikawa and Yoshioka (2019), Lwakatare et al. (2019)
2.0 test techniques	Braiek and Khomh (2020), Breck et al. (2017), Berthier et al. (2021), Haq et al. (2021), Herbold and

	Haar (2020), Nurminen et al. (2019), Riccio et al. (2020), Zhang et al. (2022)

3.3 Literature Review: Phase III – Scope of Papers

The intent of the literature review was to determine if similar research exists and where the current focus is for ML quality assurance. Table 1 shows the focus is clearly on software 2.0, which may appear valid given the rapid increase in 2.0 solutions. However, the theme that emerges is "Software 1.0 lifecycles are robust, now let's create one for 2.0". This seems an unnecessary and misguided approach. Until a complete transition is made to 2.0, both lifecycles will need to be managed, and this will likely be decades. Given that assumption, the final 24 papers were also evaluated for their usefulness to the scope of this effort: extending the QAM to support 1.0 and 2.0 life cycles simultaneously.

Model testing (QC) is not this paper's primary concern, the primary concern being answering the research question "Can a 1.0 QA architecture effectively support 2.0?". While testing is part of the answer, initially what must be evaluated is "Can the *process* do it?", and process consists of QA and QC. Table 1 shows QC has captured researchers' attention, but what of QA? The last step in the literature review process was to identify how well the final papers supported extending the QAM, and this was done by labeling the scope of each paper as either "foundational to extending the QAM" or "beneficial to understanding the current ML life cycle".

Table 2. Scope of Final Papers

Foundational to Extending the QAM	Bosch et al. (2018), Braiek and Khomh (2020), Breck et al. (2017), Herbold and Haar (2020), Jöckel et al. (2021), Revanna (2019), Haakman et al. (2021)
Beneficial to Understanding ML Life Cycle	Akkiraju et al. (2020), Ashmore et al. (2021), Berthier et al. (2021), Damyanova (2020), Haq et al. (2021), Ishikawa and Yoshioka (2019), Lenarduzzi et al. (2021), Lwakatare et al. (2019), Lwakatare et al. (2020), Mannarswamy et al. (2020), Nurminen et al. (2019), Poth et al. (2020), Riccio et al. (2020), Santhanam (2020), Serban et al. (2020), Studer et al. (2021), Zhang et al. (2022)

Examples of foundational papers are Jöckel et al. (2021) mapping 1.0 testing terminology to 2.0 as different terminologies used in the two development communities can be a major obstacle. This is foundational as it aligns with the stated QAM benefit of simplifying the meaning of quality. Breck's recommendation of a checklist for model production readiness (Breck et al, 2017) is also foundational as it touches all RDDTR phases in the QA space. Most papers, not surprisingly, are beneficial to understanding the ML life cycle. A theme that emerges from these papers is one of an ML team not exposed to traditional software engineering processes and quality practices, as discussed in Software Quality for AI (Lenarduzzi et al, 2021). Nurminen et al. (2019) propose a framework for data fault injection, one example of a plethora of research efforts aimed at ML data management.

3.4 ML Life Cycle Review

Details of the model development process cannot be shared due to intellectual property concerns, but the review of ML process documents supported what is an industry standard ML life cycle: acquire training and test data, clean and curate the data, train and test a model, deploy the model.

4. Results

While software 2.0 is rapidly disrupting the software solution space and companies are forced to support two software lifecycles, a review of the literature revealed significant research into the narrow space of ML model quality assurance in the form of data curation and model test coverage, but there is a gap relative to holistic QA for ML products consisting of software 1.0 and 2.0. Of the final 24 papers selected, six were deemed foundational to extending the QAM to support 2.0. The remaining papers were focused strictly on software 2.0 concerns and labeled beneficial to understanding the ML lifecycle. Clearly the research focus for ML products is squarely on ML models. The review of ML process documents supported what is an industry standard ML life cycle: acquire training and test data, clean and curate the data, train and test a model, deploy the model. While the terminology is different, these ML life cycle phases

are like software 1.0's RDDTR. Gathering information to determine which data to acquire, clean, and curate is a Requirements concern. Cleaning and curating data is a step necessary to begin training a model and can be seen as a Design deliverable. Training a model is Development and Test, and deploying a model is Test and Release. Each ML life cycle phase maps to software 1.0's general phases of RDDTR.

While the literature review shows a research gap relative to holistic QA solutions for ML products, the review of ML process documents and the mapping of ML life cycle to software 1.0's life cycle showed the answer to the research question is...perhaps. It is believed that extending the QAM to support ML will accomplish two things initially: elimination of two QA workflows, and QA for ML products. Elimination of two workflows is an obvious benefit, but QA for ML products, the rigorous and consistent conformance to process specifications by a machine and not by a team occasionally ignored, is an empirical benefit. As noted, model testing is not a primary concern of this proposal, but the QAM architecture inherently supports all phases necessary for developing and executing model tests (TAe, TDe, TEe, and DOe). Using the QAM to support model test development and execution is a future, attainable, concern.

5. Conclusion

The software industry is changing, rapidly. Software 1.0, traditional deterministic solutions that have been created for decades, is being disrupted by 2.0, artificial intelligence and machine learning. Quality assurance is not keeping pace with this rapid change. Research appears to be grasping the shiny 2.0 object, as there is a lack of quality assurance solutions for combined 1.0 and 2.0 products, which is a typical deployment of ML models. The Quality Assurance Machine (QAM) can fill this gap. Results of mapping the traditional software life cycle, RDDTR, to the ML life cycle, show an overlap which can be supported by extending the QAM. This effort would relieve ML software companies of the duplicative effort of supporting two lifecycles performing the same function.

References

Alamin, M. A. A., and Uddin, G., Quality Assurance Challenges for Machine Learning Software Applications during Software Development Life Cycle Phases, Available: https://arxiv.org/abs/2105.01195v2, 2021.

Al-Odeh, M. A., Smallwood, J., and Badar, M. A., A framework for implementing sustainable supply chain management, *International Journal of Advanced Operations Management,* vol. 13, no. 3, pp. 212-233, 2021.

Akkiraju, R., Sinha, V., Xu, A., Mahmud, J., Gundecha, P., Liu, Z., Liu, X., and Schumacher, J., Characterizing machine learning process: A maturity framework. In: Fahland D., Ghidini C., Becker J., Dumas M. (eds) Business Process Management, Lecture Notes in Computer Science, vol. 12168, pp. 17-31, Springer, Cham, 2020.

Ashmore, R., Calinescu, R., and Paterson, C., Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges, *ACM Computing Surveys*, vol. 54, no. 5, pp. 1–39, 2021.

Berthier, N., Sun, Y., Huang, W., Zhang, Y., Ruan, W., and Huang, X., Tutorials on Testing Neural Networks, Available: https://arxiv.org/abs/2108.01734, 2021.

Bosch, J., Olsson, H. H., and Crnkovic, I., It Takes Three to Tango: Requirement, Outcome/data, and AI Driven Development, *SiBW*, pp. 177-192, Available: http://ceur-ws.org/Vol-2305/paper14.pdf, 2018.

Braiek, H. B., and Khomh, F., On Testing Machine Learning Programs, *Journal of Systems and Software*, vol. 164, 110542, 2020.

Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D., The ML test score: A rubric for ML production readiness and technical debt reduction, *Proceedings of IEEE Big Data*, pp. 1123–1132, Available: https://research.google/pubs/pub46555, 2017.

Damyanova, B., Quality attributes in AI-ML-based systems: Differences and challenges, Available: https://doi.org/10.18419/OPUS-11309, 2020.

Defeo, J.A., *Juran's Quality Handbook: The Complete Guide to Performance Excellence*, 7th Edition, McGraw Hill, New York, 2016.

Felderer, M., and Ramler, R., Quality Assurance for AI-Based Systems: Overview and Challenges (Introduction to Interactive Session), In: Winkler D., Biffl S., Mendez D., Wimmer M., Bergsmann J. (eds) Software Quality: Future Perspectives on Software Engineering Quality, SWQD 2021, Lecture Notes in Business Information Processing, vol. 404. Springer, Cham, 2021.

Guraja, P. K., Badar, M. A., Moayed, F. A., and Kluse, C. J., Systematic Literature Review of the Impact of State Budget Cuts on Public Higher Education Institutions in the U.S., Proceedings of the 12th *International Conference on Industrial Engineering and Operations Management,* Istanbul, Turkey, March 7-10, 2022.

Haakman, M., Cruz, L., Huijgens, H., and Deursen, A. van., AI lifecycle models need to be revised, *Empirical Software Engineering*, vol. 26, article no. 95, 2021.

Haq, F. U., Shin, D., Nejati, S., and Briand, L., Can Offline Testing of Deep Neural Networks Replace Their Online Testing? *Empirical Software Engineering*, vol. 26, article no. 90, 2021.

Herbold, S. and Haar, T., Smoke Testing for Machine Learning: Simple Tests to Discover Severe Defects, Available: https://arxiv.org/abs/2009.01521, 2020.

IEEE Standard for Software Quality Assurance Processes, IEEE Std 730-2014 (Revision of IEEE Std 730-2002), 1–138, doi: 10.1109/IEEESTD.2014.6835311, 2014.

Inciteful, A better way to discover relevant literature, Available: https://inciteful.xyz/p, Accessed on January 29, 2022.

Ishikawa, F. and Yoshioka, N., How Do Engineers Perceive Difficulties in Engineering of Machine-Learning Systems? - Questionnaire Survey, *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER IP)*, pp. 2–9, 2019.

Jöckel, L., Bauer, T., Kläs, M., Hauer, M. P., and Groß, J., Towards a Common Testing Terminology for Software Engineering and Artificial Intelligence Experts, Available: https://arxiv.org/abs/2108.13837, 2021.

Khan, K. S., Kunz, R., Kleijnen, J., and Antes, G., Five steps to conducting a systematic review, *Journal of the Royal Society of Medicine*, vol. 96, no. 3, pp. 118–121, 2003.

Kneuper, R., *CMMI - Improving Software and Systems Development Processes Using Capability Maturity Model Integration*, Rocky Nook, San Rafel, CA, 2009.

Lenarduzzi, V., Lomio, F., Moreschini, S., Taibi, D., and Tamburri, D. A., Software Quality for AI: Where We Are Now? In: D. Winkler, S. Biffl, D. Mendez, M. Wimmer, and J. Bergsmann (eds.), *Software Quality: Future Perspectives on Software Engineering Quality*, vol. 404, pp. 43–53, 2021.

Lwakatare, L. E., Crnkovic, I., Rånge, E., and Bosch, J., From a Data Science Driven Process to a Continuous Delivery Process for Machine Learning Systems, In: M. Morisio, M. Torchiano, A. Jedlitschka (eds) *Product-Focused Software Process Improvement*, vol. 12562, pp. 185–201, 2020.

Lwakatare, L. E., Raj, A., Bosch, J., Olsson, H., and Crnkovic, I., A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation, In: P. Kruchten, S. Fraser, F. Coallier (eds) *Agile Processes in Software Engineering and Extreme Programming*, vol. 355, pp. 227–243, 2019.

Mannarswamy, S., Roy, S., and Chidambaram, S., Tutorial on Software Testing & Quality Assurance for Machine Learning Applications from research bench to real world, *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, pp. 373–374, 2020.

Nurminen, J. K., Halvari, T., Harviainen, J., Mylläri, J., Röyskö, A., Silvennoinen, J., and Mikkonen, T., Software Framework for Data Fault Injection to Test Machine Learning Systems, *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 294–299, 2019.

Poth, A., Meyer, B., Schlicht, P., and Riel, A., Quality Assurance for Machine Learning – an approach to function and system safeguarding, *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pp. 22–29, 2020.

Revanna, S., A Systematic Approach to extend the common Software Testing Types with modules specific to the field of Machine Learning, Master's Thesis, Institute of Software Technology, University of Stuttgart, pp. 1- 97, 2019.

Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., and Tonella, P., Testing machine learning based systems: A systematic mapping. *Empirical Software Engineering*, vol. *25*, pp. 5193–5254, 2020.

Santhanam, P., Quality Management of Machine Learning Systems, In: O. Shehory, E. Farchi, G. Barash (eds) *Engineering Dependable and Secure Machine Learning Systems*, vol. 1272, pp.1-13, 2020.

Schroeder, R.G., Linderman, K., Liedtke, C., Choo, A.S., Six Sigma: definition and underlying theory, *Journal of Operations Management*, vol. 26, no. 4, pp. 536-554, 2008.

Shafer, S.M., Moeller, S.B., The effects of Six Sigma on corporate performance: an empirical investigation, *Journal of Operations Management*, vol. 30, no. 7-8, pp. 521–532, 2012.

Serban, A., Blom, K. van der, Hoos, H., and Visser, J., Adoption and Effects of Software Engineering Best Practices in Machine Learning, *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020.

Sherin, S., Khan, M. U., and Iqbal, M. Z., A Systematic Mapping Study on Testing of Machine Learning Programs, Available: https://arxiv.org/pdf/1907.09427v1, 2019.

Siebert, J., Joeckel, L., Heidrich, J., Nakamichi, K., Ohashi, K., Namba, I., Yamamoto, R., and Aoyama, M., Towards Guidelines for Assessing Qualities of Machine Learning Systems, In: Shepperd M., Brito e Abreu F., Rodrigues da Silva A., Pérez-Castillo R. (eds) *Quality of Information and Communications*

Technology. QUATIC 2020. Communications in Computer and Information Science, vol. 1266, pp. 17-31. Springer, Cham, 2020.

Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., and Müller, K.-R., Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology, *Machine Learning and Knowledge Extraction,* vol. 3, no. 2, pp. 392-413, 2021.

Zhang, J. M., Harman, M., Ma, L., and Liu, Y., Machine Learning Testing: Survey, Landscapes and Horizons, *IEEE Transactions on Software Engineering,* vol. 48, no. 1, pp. 1-36, 2022.

Biographies

Shane Downing is a PhD student at Indiana State University and a software quality assurance manager with more than two decades experience in the software industry. He has contributed to the software quality space as a test engineer, test architect, and manager. He received BSc degrees in Automated Systems Engineering Technology and Computer Science, and an ME in Embedded Systems Engineering.

M. Affan Badar, PhD, CPEM, IEOM Fellow is a Professor in the Department of Applied Engineering and Technology Management at Indiana State University, USA. From 2016 to 2018, he was a professor and chair of the Department of Industrial Engineering and Engineering Management at the University of Sharjah, UAE. He received a Ph.D. in Industrial Engineering from the University of Oklahoma, MS in Mechanical Engineering from King Fahd University of Petroleum and Minerals, MSc in Industrial Engineering and BSc (Hons) in Mechanical Engineering from Aligarh Muslim University.