Electronic Communications of the EASST
Volume 82 (2022)

11th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation

-

Doctoral Symposium, 2022

Software Engineering meets Artificial Intelligence

Holger Klus, Christoph Knieke, Andreas Rausch and Stefan Wittek

16 pages

Guest Editors: Sven Jörges, Salim Saay, Steven Smyth
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Software Engineering meets Artificial Intelligence

Holger Klus[1], **Christoph Knieke**[2], **Andreas Rausch**[3] **and Stefan Wittek**[4]

[1] holger.klus@iu.org
IU International University of Applied Science
[2] christoph.knieke@tu-clausthal.de
[3] andreas.rausch@tu-clausthal.de
[4] stefan.wittek@tu-clausthal.de
Institute for Software and Systems Engineering
Technische Universität Clausthal, Clausthal-Zellerfeld, Germany

Abstract: With the increasing use of AI in classic software systems, two worlds are coming closer and closer to each other that were previously rather alien to each other, namely the established discipline of software engineering and the world of AI. On the one hand, there are the data scientists, who try to extract as many insights as possible from the data using various tools, a lot of freedom and creativity. On the other hand, the software engineers, who have learned over years and decades to deliver the highest quality software possible and to manage release statuses. When developing software systems that include AI components, these worlds collide. This article shows which aspects come into play here, which problems can occur, and how solutions to these problems might look like. Beyond that, software engineering itself can benefit from the use of AI methods. Thus, we will also look at the emerging research area AI for software engineering.

Keywords: Artificial Intelligence, Software Engineering

1 Introduction

1.1 Motivation

In more and more software projects, solutions are being solved in whole or in part using AI approaches. This creates many new challenges in different aspects of development. These include technical challenges, but also organizational and legal aspects have to be considered, among others.

In this paper we will show the different aspects that a software engineer has to master to integrate AI-based technologies. At this, we share our practical experience from industry projects in developing and operating AI-based software in order to help others involved in such projects. When we talk about AI in the following, we primarily mean machine learning based approaches.

1.2 Contribution

This paper is an experience paper based on the different experiences the authors have gained in the industry. They were gathered from software development projects in which mass data was analyzed using AI support. Among other things, software for testing the integrity of industrial plants was developed in this context, and the results that the software delivers are thus safety-relevant. The authors were involved in these projects both as leading software architects and in leading management roles and were thus jointly responsible for the technical and organizational implementation of AI-based, safety-critical projects. In this experience report, the experience gained in the course of these projects has been incorporated, among other things, and is intended to provide readers with guidance for similar projects.

1.3 Related Work

In recent years, there have been several survey papers examining the challenges and current research approaches in the field of software engineering for AI (SE4AI) and vice versa - AI for software engineering (AI4SE) - and highlighting possible synergies between the two research areas. In the following, an overview on the current state of research is given on this field. The first three studies [FEG+20, Gir21, MBF+22] focus on SE4AI whereas the studies [SMME21, PSC+20] investigate the AI4SE research field.

The study by Giray [Gir21] aims at systematically identify, analyze, summarize, and synthesize the current state of software engineering research for engineering ML systems. Therefore, the author performs a systematic literature review revealing that none of the SE areas (e.g., testing) have a mature set of tools and techniques. In addition, Giray concludes that reporting on lessons learned from action research in industry can provide valuable insights for answering research questions.

Fischer et al. [FEG+20] discuss key challenges and lessons learned from past and ongoing research along the development cycle of machine learning systems. They regard hurdles and challenges from current machine learning paradigms, and key challenges of AI model/system lifecycle. They uncover a fundamental theory-practice gap in machine learning with impact on reproducibilty and stability.

Martínez-Fernández et al. [MBF+22] conduct a systematic mapping study to collect and analyze state-of-the-art knowledge about SE for AI-based systems. Software testing and software quality are the most prevalent in their sample. Software construction, software requirements, and software maintenance are less represented revealing a research gap. Moreover, only few holistic approaches exist, indicating that the research field SE for AI is still in an early stage.

Shafiq et al. [SMME21] conduct a literature review of using machine learning in software development life cycle stages and investigate the relation of software development life cycle stages with ML types, tools, and techniques. They also reveal open areas of research (e.g., maintenance) and propose future research directions (e.g., requirements traceability).

The study by Perkusich et al. [PSC+20] focus on the specific field of agile software development (ASD) and synthesize and analyze the state-of-the-art of the field of applying intelligent techniques to ASD. They refer to "intelligent techniques" as "*a technique that explores data (from digital artifacts or domain experts) for knowledge discovery, reasoning, learning, plan-*

ning, natural language processing, perception or supporting decision-making." The survey reveals an increase in the number of studies applying intelligent techniques to ASD. In addition, machine learning, reasoning under uncertainty, and search-based solutions are the most used intelligent techniques in the context of ASD.

2 Software Engineering for AI: Challenges

Based on our experience, we have identified six key challenges in developing AI-based software systems, which are shown in Figure 1.

Figure 1: Software Engineering for AI: Challenges

Here, we highlight both cultural and technical aspects. In the following sections, we will now go into each of these aspects in more detail.

2.1 Expectation Management

2.1.1 Context

Expectations of the possibilities offered by artificial intelligence have risen steadily in recent years. Companies are expecting new business models, opportunities to increase efficiency and effectiveness, better marketing opportunities for their products and services, and much more. And the numerous success stories that can be read about are fuelling this high expectation more and more [Mak17].

2.1.2 Experience and Observations

Decision-makers are being led to believe that embedding AI in software solutions does not pose any significant hurdles. Furthermore, many AI technologies are in the middle of the Peak of

Inflated Expectations of the well known Gartner Hype Cycle [LF03]. However, the maturity of AI frameworks themselves is limited [HH22].

Transparent communication is a key factor here in order to prevent false expectations from the outset. Software architects have a special role to play here. One of the reasons for this is that software architects are already assigned a mediating role between the most diverse stakeholders involved in a project [Gel20]. Experienced software architects have experience in discussing technical issues and the consequences of technical decisions with different stakeholders of a project and a company in a way that is appropriate for the target group. This ability is also particularly in demand when it comes to managing expectations when introducing AI as a component of software systems. This already starts in the early decision-making phase, when it is first a matter of evaluating whether the use of AI makes sense.

The integration of AI into software systems and processes has numerous consequences. This is because hybrid systems of this kind require, among other things, a modified management of the software lifecycle [LC16]. Here, not only the lifecycle of the source code, but also of the AI model with all its special features must be managed. Users may need to be prepared for the system to behave in ways that are not deterministic or unanticipated by them. For example, if the underlying AI model is adapted, whether explicitly from the outside or by a self-learning mechanism, the behavior of the system will also change. This in turn can jeopardize acceptance and trust in such hybrid systems.

2.1.3 Lessons Learned

It is important to actively manage expectations with regard to AI from the outset and to highlight which foundations must be laid for the successful integration of AI into software systems and thus into business processes, and what consequences arise from this, like the need to adapt the whole software lifecycle management. Moreover, the architect's role in managing expectations regarding AI should be clearly defined from the outset.

2.2 Clash of Cultures

The use of Data Science in classical software development projects has increased in recent years, and so has the need for collaboration between the disciplines of Data Science and classical software development [LC16, KZDB16]. In this section, we describe our experience on this aspect.

2.2.1 Context

In the following, we will first outline the initial situation on which our experience is based and afterwards will describe our experience.

The scenario considered here is based on a business model in which sensor data is gathered and analyzed on a large scale on behalf of customers. In the course of this evaluation, certain anomalies in the data must be detected with the highest possible reliability. If certain anomalies in the data remain undetected, human lives can be endangered in the worst case.

The evaluation is partially automated. In this process, the data is first preprocessed and then evaluated in detail by data evaluators. The detected anomalies are reported to the end customer

including relevant additional information. The data evaluators are usually not data scientists, but have essentially been trained to recognize and evaluate specific, predefined anomalies in the data with the help of domain-specific knowledge.

To support the evaluation process of the sensor data, a software development team has been (further) developing and operating an extensive software suite for many years. The team consists of the usual roles in software engineering, such as software developers for front-end and back-end, testers, usability engineers, software architects, release managers and requirements engineers.

The team delivers a release to a total of more than one thousand users at regular intervals of a few months and on demand. The most important quality criteria here include reliability in anomaly detection, performance and scalability, maintainability of the software, and traceability of the evaluation results.

In addition to software development, there are other departments that deal with new, forward-looking possibilities for data evaluation and optimization. Here, data scientists, physicists and mathematicians are constantly working on new solutions and ideas.

Within this framework, prototypes and new algorithms are constantly being developed and tested with selected users. AI naturally plays an important role in this context. The whole organizational setup is similar to what is introduced in [NZLK22], where also the clash of cultures has been covered.

Based on these research initiatives, it was decided to integrate the possibilities of AI more and more into the standard process of data evaluation. In addition to the productive evaluation suite, a prototypical evaluation suite was created, which evaluates the data to a large extent with the help of AI. The software development team was not initially involved in this development. The prototype was gradually made available to selected evaluators by the Data Scientists in parallel with the productive environment. As the number of circulating versions and variants of the prototype increased, they came under more and more pressure. It was then decided to integrate parts of the research departments into the team of professional software developers and thus integrate AI into the productive environment in the medium term.

2.2.2 Experience and Observations

In the course of the process described above, the following main experiences were made:

- From the Data Scientists' point of view, the software development process was perceived as very heavyweight and rule-bound. The environment was not one in which Data Scientists felt comfortable.
- From the software development team's perspective, the Data Scientists' approach to testing, usability, release management, and maintainability was perceived as unprofessional due to its experimental approach.
- As part of the parallel further development of both the prototype and the productive environment, mutual skepticism and reservations grew stronger over the months.
- The selection of libraries used during prototype development was later questioned in part by the software development team because, in the view of the software developers, some

of them did not meet the quality requirements of a productive environment.

- A later consolidation was only possible by removing numerous hurdles.

Based on these experiences and observations, the lessons learned are summarized in the following section.

2.2.3 Lessons Learned

The lessons learned can be summarized as follows:

- The working methods and focus of Data Scientists on the one hand and software developers on the other hand differ significantly.
- Data Scientists are used to experiment with data and algorithms. They have a high expertise in data science and a deep understanding of the algorithms. They also focus on short release cycles. Aspects like usability, release management or longtime support and maintenance have a rather subordinate role.
- Software developers focus on long-term maintainability, and quality aspects in general. They have a lot of experience in maintaining and operating a complex software system over many years. Aspects such as experimental procedures or spontaneous releases without adherence to predefined processes play a rather subordinate role.
- In projects where it is foreseeable that both software engineering and data science skills are needed, combine all required roles in the team from the outset, e.g., by delegating individual data scientists to the development team.
- Actively promote mutual understanding through joint appointments, workshops, or joint conference visits.

2.3 Requirements Engineering and System Design

Requirements are collected, documented and managed as part of the Requirements Engineering. These requirements form the basis for all further steps in the software engineering process, regardless of which process model is followed [Som07].

2.3.1 Context

In classic software development, solution strategies are developed from functional and non-functional requirements. This results in a system design, consisting of components and subcomponents, as well as their relationships to each other. Based on this, source code is developed, tested and delivered. In the classic waterfall model, this process is only run through once, in agile methods this happens iteratively and incrementally [Som07]. This basic process is shown in simplified form in Figure 2.

Whether the functional and non-functional requirements have actually been met depends primarily on the skills of the architects, developers, testers and other stakeholders involved. At

Figure 2: Classical engineering approach

runtime input data is processed and converted into outputs as shown schematically in Figure 3. The outputs can be used to check whether the requirements have been met.

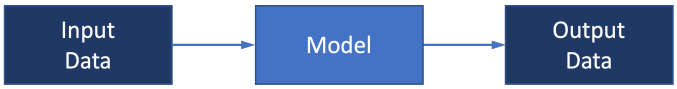

Figure 3: Abstract runtime view on a component or system

In AI-based applications, the relationship shown schematically in Figure 3 still applies. However, the creation of the model is different. Here, the entire system or one or more of its subcomponents are trained using machine learning approaches. Among other things, a distinction can be made between supervised and unsupervised learning strategies. All these approaches are based on data sets [Ber21]. The functional and non-functional properties of the system here no longer depend only on the skills of the stakeholders involved, but also largely on the amount, selection and quality of the data with which the system or its subcomponents were trained via the input data/output data shown in Figure 4.

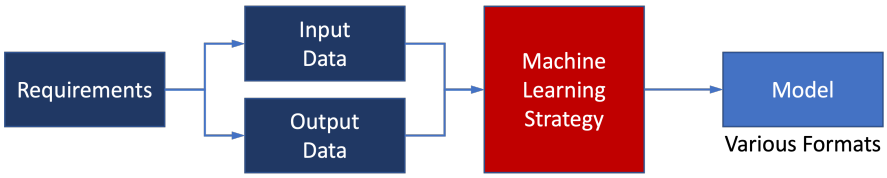

Figure 4: Development view on an AI component

2.3.2 Experiences and Observations

The development view on an AI component as shown in Figure 3 has implications for both development and the system at runtime:

- For the development phase, this means that in addition to the usual tools and libraries, machine learning is now available as a further tool to implement requirements in the best possible way.
- While development teams have already gained experience over many years with regard to the implications of using certain technologies such as programming libraries, databases and so on, these empirical values for AI-based systems or components are not yet available

to the same extent. And so the effects of their usage on the functional and non-functional characteristics can be estimated more difficult.

- Since this largely depends on the amount, selection and quality of the data, these aspects must also be considered, while in classic software development they were mostly irrelevant for the development and had no impact on the quality of the software itself.

- Belani et al. [BVC19] identify for each requirements engineering activity (e.g., elicitation, documentation, validation) the challenges w.r.t. the AI-related entities (data, model, system). For validation, e.g., they see "training data critical analysis" and "various data dependencies" as challenges regarding AI-related entity *data*. This RE4AI taxonomy specified in [BVC19] and the challenges highlighted are helpful in practice for the tailoring of development process.

2.3.3 Lessons Learned

To summarize, during the operation of AI-based software, in addition to the usual maintenance work and release management, there are now additional tasks that regard the management of training and validation data. Appropriate mechanisms must be established here to ensure, among other things, that it is possible to determine which software and which model was trained with which data and thus enabling traceability [BVC19].

2.4 Quality Assurance

2.4.1 Context

A still widespread process model in software engineering is the concept of V-shaped process models, at least in the field of functional safety or more generally of technical systems, especially embedded systems (e.g., in the automotive domain [LZZ16]), and is also frequently called for in corresponding standards due to its emphasis on verification and validation [Kne18]. In this section, we focus on the field of safety critical systems, developed by V-shaped process models.

Similar to the waterfall process model, the V-Model organizes the software development process in phases [MM10]. In addition to these development phases, the V-Model also defines the procedure for quality assurance (testing) by contrasting the individual development phases with test phases. On the left, the process begins with a functional/technical specification, which is expanded in ever greater detail to a technical specification and implementation basis. At the bottom, the implementation takes place, which is then tested on the right-hand side against the corresponding specifications on the left-hand side [MM10]. This creates the figurative "V" that gives the project its name, which contrasts the individual development levels with their respective test levels.

2.4.2 Experiences and Observations

- In practice, there are frequent adaptations of the AI-based model, e.g., due to new training data. The model must be tested again after each adaptation. The tests of the AI-based

models can have long execution times and are thus very time-consuming. This was also stated in [Gir21].

- In addition, the handling of the many model versions proves to be challenging, described, e.g., in [Gir21].

- If the AI-Model features online learning [HK93], it is continuously improved during operation by incorporating feedback or new data. At the same time, new model versions are created at development time, but these differ as that runtime improvements is missing here. This inevitably leads to a drifting apart of the model versions in operation and in development. The authors observed in industrial projects developed based on the V-Model, that the handling of these versions, the subsequent merging and the quality assurance turn out to be difficult here.

- As described in Section 2.3 the functional requirements of AI systems based on machine learning are often specified by a finite set of input output examples. In addition, many current AI algorithms have a black-box nature. As a result, many formal quality assurance methods such as verification can not be employable on the AI components of the systems since there is not even an explicit specification of the desired functionality. The only way to do QA here is statistic testing using again input output examples of data. This is problematic in the context of dependable systems in safety critical applications.

2.4.3 Dependability Cage Approach for AI-based Components

In [AGR+18] we presented the vision of a holistic, iterative software development approach for dependable autonomous systems (see Figure 5) by closing the gap between runtime and development time models. The development part of this approach is based on the V-Model. In operation, an autonomous system is repeatedly confronted with situations that were not considered at development time. For this reason, the system has to adapt through adjustment and learning in order to react to the changed environment.

However, this means that the system could act outside its specification. In order to identify these changes and to consider them in the further development cycles of the system, we have introduced the concept of "Dependability Cages". It examines at runtime the behavior of the system and its environment against boundary conditions defined and tested at development time. In unforeseen situations, Dependability Cages can not only intervene in the configuration of the system, but also record the new situations and feed them into the iterative development process as new development artifacts. This approach is particularly suitable for monitoring AI-based components and assuring system safety at runtime.

2.5 Tool Chain and Processes

2.5.1 Context

For the development and operation of classical software systems as well as for AI applications, a variety of tools exist today. In the case of classic software systems, professional tools can be

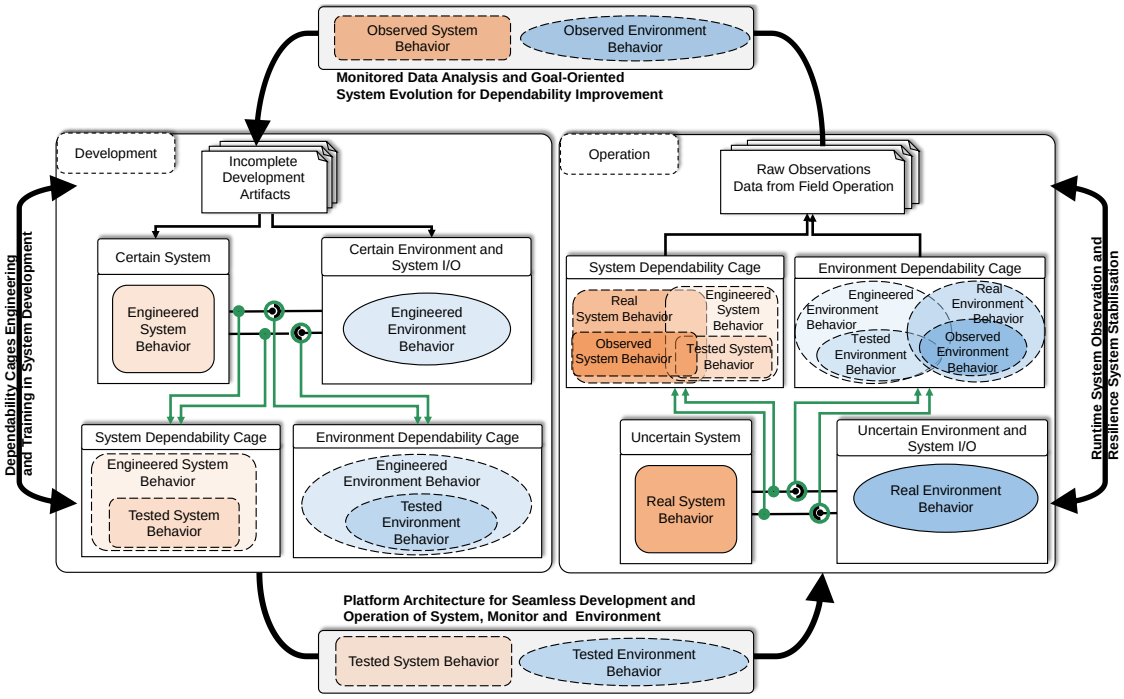

Figure 5: Dependability Cage Approach [AGR+18]

used, which can also be integrated into a seamless tool chain. On the side of AI applications, too, there are now numerous mature tools for development and operation available [FEG+20].

2.5.2 Experiences and Observations

One challenge, however, is that different tools are used for AI applications than for classic software systems. In practice, this heterogeneity turns out to be a major obstacle [KZDB17], complicating many tasks of the engineer, such as release management and quality assurance. Integrating tools for AI aspects into traditional software development infrastructure is difficult.

Several approaches now exist that aim to provide end-to-end tool support for AI applications by adding AI aspects to existing IDEs [Gir21], such as Azure ML for Visual Studio Code [ABB+19]. Currently, there are several tools that consider different requirements regarding AI development, such as visualization and comparison of experiments. However, when considering the approaches, it is still important to note that these are essentially research works, and the prototypes are yet to be developed into professional tools to be used in practice.

Further challenges arise when AI components are to be used on real-time systems. Here, there are generally limited resources, such as with embedded systems. Inference very often has to deal with real-time constraints, tight power or energy budgets, and security threats [FEG+20].

In the development of variant-rich systems, such as in the automotive sector, software product line engineering methods are used today to manage complexity and achieve the highest possible degree of reuse. For systems with AI components, it is unclear how the different versions and

variants of the ready-made, pre-trained networks can be managed and integrated into classic software product line engineering.

2.5.3 Lessons Learned

To sum up, the aspects just described make it clear that the development of tool chains for software systems with AI aspects is a major challenge. On the one hand, the software architect must be able to select the appropriate and practice-ready tools in the rapidly changing field of tool environments for AI and integrate them into a tool chain with the classical software development infrastructure. On the other hand, the development processes must be coordinated and architecture constraints must be taken into account.

2.6 Trust, Legal and Ethics

2.6.1 Context

The advance of AI has far-reaching consequences for our lives. It is already foreseeable that in a few years many jobs will be replaced by AI. Studies here assume up to 30% displacement of jobs by the mid-2030's [BGS+17]. On the other hand, we are confronted with AI applications in more and more areas, such as autonomous driving [NMS22], AI in healthcare (analysis, presentation, and comprehension of complex medical and health care data) [YBK18], chart robots [WGKD18], and surveillance systems [FHCL22]. This often involves collecting data from users, and it remains opaque what is done with that data. Seen from the outside, AI models are like a black box whose inner workings remain hidden from the user. Decisions made by the AI cannot be traced. Predictability and reliability of AI technology, as well as understanding how the technology works and the intentions of its creators, are essential factors for trust. Especially for critical applications, the user wants to understand the reasons for classification and know under which conditions the system is trustworthy and when it is not. Explainability is quite essential in building trust between an AI system and its users.

One area of research that deals with this issue is Explainable AI [XUD+19]. This area is often divided into three subcategories: Understandable AI (understanding the model upon which the AI decision-making is based), Reasonable AI (understand the reasons behind each individual prediction), and Traceable AI (trace the prediction process from math algorithm to data). Different techniques exist in the field of explainable AI, e.g., model agnostic techniques [HSM+22].

2.6.2 Experiences and Observations

The software architect must have an understanding of the various techniques of Explainable AI so that he/she is able to select the appropriate techniques for the respective project and make them transparent to the development team. For this purpose, the architecture must also be considered, since it has an important influence on the selection and application of the techniques. Furthermore, the architect acts as a mediator between the management, the customer and the development team involving the Explainable AI techniques, and has to educate if necessary.

There are also efforts to identify negative impacts and possible risks of AI-based systems [Gir21] . For instance, "Ethics guidelines for trustworthy AI" define the ethical principles that an

ML system should adhere to [AI 19]. These principles should be continuously considered while engineering AI systems to ensure Trustworthy AI.

2.6.3 Lessons Learned

In summary, we can state that users often do not have confidence in the correctness of the decisions of AI-based systems at the beginning. In practice, we have found that we can improve user confidence in AI by proceeding as follows: Initially, the AI only offers suggestions, but does not make any decisions itself. After the AI proved that it reliably makes good decisions, we then gradually scaled back the classic approach and increasingly let the AI make the decisions. This gradual change helps build trust in AI-based systems.

3 AI for Software Engineering

Modern software systems are highly complex and have a long lifetime. During the engineering of such systems, enormous amounts of data are generated over time. This includes the development artifacts (e.g., source code, models, documentation of design decisions) in different versions and variants, but also the data acquired during operation. All this data offers great potential to support software engineering. Appropriate data-based engineering processes and techniques can ensure a managed evolution of systems, e.g., through DevOps approaches [LRK+19] that combine development (Dev) and operations (Ops).

The emerging field of "AI for Software Engineering" (AI4SE) addresses this topic and proposes AI-based methods and tools based on this data, e.g., to support and even partially or completely automate development decisions. Many software engineering tasks can be formulated as data analysis tasks and thus can be supported, e.g., with AI algorithms. Major drivers for the field of AI4SE are the growing size and complexity of software systems increasing the degree of difficulty of the software engineering tasks [MBF+22]. In general, AI4SE can be referred to as intelligent software engineering and a portfolio of software engineering techniques, which "*explore data (from digital artifacts or domain experts) for knowledge discovery, reasoning, learning, planning, natural language processing, perception or supporting decision-making*" [PSC+20]. Xie [Xie18] analyzes synergies between AI and software engineering and formulates open research questions:

- how to define or determine levels of intelligence in software engineering solutions,
- how to bring high levels of intelligence in software engineering solutions, and
- how to synergically integrate machine intelligence and human intelligence (e.g., domain knowledge or insight) to effectively tackle challenging software engineering problems.

An emerging field of AI4SE are programming assistants improving developer productivity through code synthesis, code search, or other types of code recommender systems. A widespread programming assistant is Codex [Ope23], a GPT language model fine-tuned on publicly available code from GitHub to produce code given a natural language description. Chen et al. [CTJ+21] evaluate the capabilities of Codex. While Codex has the potential to be useful in a range of ways,

e.g., enable non-programmers to write specifications, it has also several limitations. They find, e.g., that Codex is not sample efficient to train, and they explore prompts on which Codex is likely to fail or display counter-intuitive behavior.

In 2021, GitHub and OpenAI introduced GitHub Copilot [Git23], an "AI pair programmer" for JetBrains IDEs, and Visual Studio Code Neovim, powered by the Codex model. Given a natural language description of the target functionality, GitHub Copilot can generate corresponding code in several programming languages. The empirical study in [NN22] analysis the correctness and understandability of GitHub Copilot's synthesized code solutions based on 33 LeetCode questions. They find that Copilot's Java suggestions have the highest correctness score (57%) while JavaScript is lowest (27%). Potential shortcomings are identified, e.g., such as generating code that can be further simplified/compacted [NN22].

4 Conclusion

Due to the increasing integration of AI technologies into traditional software systems, the worlds of data science and software engineering are colliding. This creates several challenges for software engineers, including expectation management, handling cultural clashes between different backgrounds, requirements engineering, quality assurance, tool chains and processes, and trust, legal, and ethical issues. Within this paper we present our perspectives on this challenges based on the experience in several industry project within this clash of cultures.

To successfully integrate AI into software systems, having sufficient and quality data is crucial. However, small and medium-sized companies often lack the basic foundation to use it productively. Communication between stakeholders is essential to prevent false expectations, and the decision to use AI should be conscious and based on technical and business perspectives. Software architects play a key role in managing expectations and ensuring that both worlds, software engineering, and AI, understand each other's priorities.

Monitoring AI-based components is essential to identify changes and intervene when necessary. Dependability cages provide an architectural framework for this task.

The heterogeneity of AI tools and traditional software development infrastructure complicates release management and quality assurance, with developing tool chains for software systems with AI aspects being a significant challenge.

Understanding AI models and their decisions is crucial for building trust in AI-based systems, with explainable AI techniques aiming to make AI more transparent to users. Ethics guidelines and identifying negative impacts of AI-based systems are important for trustworthy AI.

Finally, AI4SE proposes using data generated during the engineering of complex software systems to support and automate development decisions, driven by the growing size and complexity of software systems. Many on this applications remain promising and are an active field of research.

In our ongoing research, we have projects addressing SE4AI as well as AI4SE. In the AI4SE field, we are, e.g., developing a learning based approach for specification completion by observing traces at runtime. In addition, we work on an approach for AI-based failure analysis during real-time validation of automotive software systems. As a future work regarding SE4AI we plan to conduct further experiments in concrete projects focusing on qualitative analysis on

the experiences discussed in this paper.

Bibliography

[ABB+19] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, T. Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Pp. 291–300. 2019.

[AGR+18] A. Aniculaesei, J. Grieser, A. Rausch, K. Rehfeldt, T. Warnecke. Towards a holistic software systems engineering approach for dependable autonomous systems. In *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*. Pp. 23–30. 2018.

[AI 19] AI HLEG. High-level expert group on artificial intelligence. *Ethics guidelines for trustworthy AI*, p. 6, 2019.

[Ber21] E. Bernard. *Introduction to Machine Learning*. Wolfram Media Inc, 2021.

[BGS+17] J. Brown, T. Gosling, B. Sethi, B. Sheppard, C. Stubbings, J. Sviokla, J. Williams, D. Zarubina, L. Fisher. Workforce of the future: The competing forces shaping 2030. *London: PWC*, 2017.

[BVC19] H. Belani, M. Vukovic, Ž. Car. Requirements engineering challenges in building AI-based complex systems. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. Pp. 252–255. 2019.

[CTJ+21] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[FEG+20] L. Fischer, L. Ehrlinger, V. Geist, R. Ramler, F. Sobiezky, W. Zellinger, D. Brunner, M. Kumar, B. Moser. Ai system engineering—key challenges and lessons learned. *Machine Learning and Knowledge Extraction* 3(1):56–83, 2020.

[FHCL22] C. Fontes, E. Hohma, C. C. Corrigan, C. Lütge. AI-powered public surveillance systems: why we (might) need them and how we want them. *Technology in Society* 71:102137, 2022.

[Gel20] C. Gellweiler. Types of IT Architects: A Content Analysis on Tasks and Skills. *Journal of theoretical and applied electronic commerce research* 15(2):15–37, 2020.
doi:10.4067/S0718-18762020000200103

[Gir21] G. Giray. A software engineering perspective on engineering machine learning systems: State of the art and challenges. *Journal of Systems and Software* 180:111031, 2021.

[Git23] GitHub, Inc. GitHub Copilot - Your AI pair programmer. 2023. https://copilot.github.com

[HH22] S. Herbold, T. Haar. Smoke testing for machine learning: simple tests to discover severe bugs. *Empirical Software Engineering* 27(2):45, 2022.

[HK93] T. M. Heskes, B. Kappen. On-line learning processes in artificial neural networks. In *North-Holland Mathematical Library*. Volume 51, pp. 199–233. Elsevier, 1993.

[HSM+22] A. Holzinger, A. Saranti, C. Molnar, P. Biecek, W. Samek. Explainable AI methods-a brief overview. In *xxAI-Beyond Explainable AI: International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers*. Pp. 13–38. 2022.

[Kne18] R. Kneuper. Die geschichtliche Entwicklung des V-Modells. Technical report, IUBH Discussion Papers-IT & Engineering, 2018.

[KZDB16] M. Kim, T. Zimmermann, R. DeLine, A. Begel. The Emerging Role of Data Scientists on Software Development Teams. In *Proceedings of the 38th International Conference on Software Engineering*. ICSE '16, pp. 96–107. Association for Computing Machinery, New York, NY, USA, 2016.

[KZDB17] M. Kim, T. Zimmermann, R. DeLine, A. Begel. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering* 44(11):1024–1038, 2017.

[LC16] D. Larson, V. Chang. A review and future direction of agile, business intelligence, analytics and data science. *International Journal of Information Management* 36(5):700–710, 2016.

[LF03] A. Linden, J. Fenn. Understanding Gartner's hype cycles. *Strategic Analysis Report Nº R-20-1971. Gartner, Inc* 88:1423, 2003.

[LRK+19] L. Leite, C. Rocha, F. Kon, D. Milojicic, P. Meirelles. A survey of DevOps concepts and challenges. *ACM Computing Surveys (CSUR)* 52(6):1–35, 2019.

[LZZ16] B. Liu, H. Zhang, S. Zhu. An incremental V-model process for automotive development. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. Pp. 225–232. 2016.

[Mak17] S. Makridakis. The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms. *Futures* 90:46–60, 2017.

[MBF+22] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, S. Wagner. Software engineering for AI-based systems: a survey. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31(2):1–59, 2022.

[MM10] S. Mathur, S. Malik. Advancements in the V-Model. *International Journal of Computer Applications* 1(12):29–34, 2010.

[NMS22] B. Namatherdhala, N. Mazher, G. K. Sriram. Uses of Artificial Intelligence in Autonomous Driving and V2X communication. *International Research Journal of Modernization in Engineering Technology and Science* 4(7):1932–1936, 2022.

[NN22] N. Nguyen, S. Nadi. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*. Pp. 1–5. 2022.

[NZLK22] N. Nahar, S. Zhou, G. Lewis, C. Kästner. Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process. In *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22, p. 413–425. Association for Computing Machinery, New York, NY, USA, 2022.
doi:10.1145/3510003.3510209
https://doi.org/10.1145/3510003.3510209

[Ope23] OpenAI. OpenAI Codex. 2023.
https://openai.com/blog/openai-codex

[PSC+20] M. Perkusich, L. C. e Silva, A. Costa, F. Ramos, R. Saraiva, A. Freire, E. Dilorenzo, E. Dantas, D. Santos, K. Gorgônio et al. Intelligent software engineering in the context of agile software development: A systematic literature review. *Information and Software Technology* 119:106241, 2020.

[SMME21] S. Shafiq, A. Mashkoor, C. Mayr-Dorn, A. Egyed. A literature review of using machine learning in software development life cycle stages. *IEEE Access* 9:140896–140920, 2021.

[Som07] I. Sommerville. *Sommerville: Software Engineering*. Pearson Studium, 2007.

[WGKD18] S. Wailthare, T. Gaikwad, K. Khadse, P. Dubey. Artificial intelligence based chatbot. *Artificial Intelligence* 5(03), 2018.

[Xie18] T. Xie. Intelligent software engineering: Synergy between AI and software engineering. In *Proceedings of the 11th Innovations in Software Engineering Conference*. Pp. 1–1. 2018.

[XUD+19] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, J. Zhu. Explainable AI: A brief survey on history, research areas, approaches and challenges. In *Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8*. Pp. 563–574. 2019.

[YBK18] K.-H. Yu, A. L. Beam, I. S. Kohane. Artificial intelligence in healthcare. *Nature biomedical engineering* 2(10):719–731, 2018.