Baština Akademije nauka i umjetnosti Bosne i Hercegovine

Artificial Intelligence in Industry 4.0: The future that comes true: AI

Karabegović, Isak; editor

2024-09-17

https://bastina.anubih.ba/handle/123456789/791

Preuzeto s Baštine Akademije nauka i umjetnosti Bosne i Hercegovine

https://bastina.anubih.ba/

DOI: 10.5644/PI2024.215.01

Influence of Artificial Intelligence on Methodologies and Processes for Engineering Software-Enabled Systems in Industry 4.0

Jasmin Jahić*[1]

Abstract: *Software currently presents is a corner stone of systems in Industry 4.0 (I4.0). To engineer software for these systems, engineers follow different methodologies and processes. These methodologies and processes aim to systemise production of high-quality software systems and make it possible to reproduce success in software engineering projects.*
With the introduction of AI in software engineering, actions that engineers perform are changing. Consequently, challenges and responsibilities of developers change. It becomes valid to ask: how will processes and methodologies in software engineering change with the introduction of AI? That means, what will be the new challenges that software engineering methodologies and processes need to solve, and which current challenges will simply disappear or become irrelevant. To answer these questions, in this paper, we abstract and summarise actions and aims of processes and methodologies in software engineering. We make predictions of what is it that humans bring to the table when it comes to software engineering, where can AI assist humans, and where AI has potential to fully replace humans. We discuss these predictions in the context of quality properties of I4.0 systems (e.g., security, safety), which must be taken into account when engineering I4.0 software-enabled systems.

Keywords: *Industry 4.0, AI, software engineering, software architecture, methodologies, processes*

1. Introduction

Industry 4.0 aims towards manufacturing flexibility, increased efficiency, and higher productivity[1]. To achieve these, it relies on communication, information, and intelligence technologies. The corner stone and the key enabler of all these technologies is software.

Engineering of software system is not an easy task. It involves many more actions than simply writing source code (Figure 1). To solve any problem using software, it is first necessary to solve it on a logical level. Programming languages, and software libraries serve to express logical solutions using source

*[1]U*niversity of Cambridge, UK*
E-mail: *jj542@cam.ac.uk*

code. Scaling the development of software solutions is a management and soft-skills challenge in software engineering. To produce solution faster and with adequate quality, it is necessary to organise development using development methodologies (e.g., as one would organise work in factories to increase throughput). These methodologies facilitate productivity, collaboration, and help to create high-quality software that fulfils its business goals. These methodologies are not of much use if they are not supported by proper processes and infrastructure. These days, source code version control, DevOps, and other infrastructure support development methodologies. But with time, software grows and it becomes hard to manage it, in particular when it comes to introducing new features. This is especially the case when software, besides providing desired functionality, needs to meet explicitly defined quality requirements (e.g., security, safety). To deal with these challenges, architects engage into system-level engineering activities. Software system architecture enhances communication, provides guidance, and serves to limit solution space for developers to ensure that software will meet its quality properties. Finally, all the activities that are performed in a software engineering project serve to fulfil business goals. These goals are not necessarily related to financial gains, but also can be centred around philanthropy or solving societal problems. As we can conclude from this overview of actions in software engineering, creating and extending software-enabled systems is a very complex undertaking. Adding AI into software engineering has potential to make it easier to deal with some of the challenges associated with presented activities andchange the way how we create software systems. However, at this point, it is still not clear what the extent of those changes will be. In this paper, we focus on processes and methodologies in software engineering impacted by adoption of AI and try to make our predictions considering engineering of I4.0 systems.

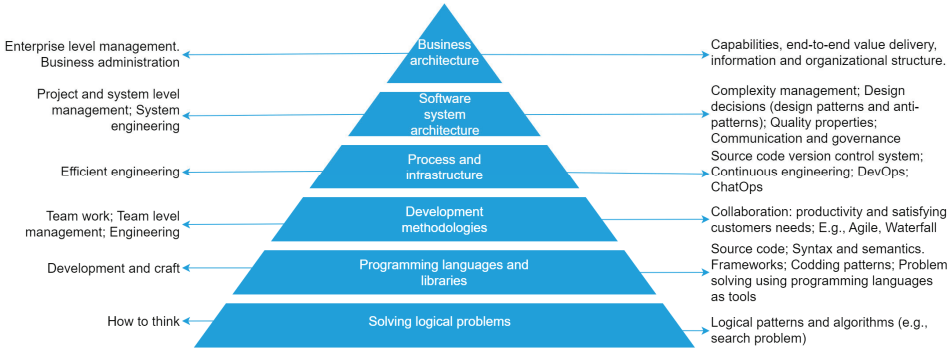

Figure 1: Actions involved in creating software-enabled systems

Besides guaranteeing that software meets its functional requirements, software engineers have to also ensure that their software-enabled systems meet quality

requirements. Depending on the system type, industry, number of users, context of deployment, and many other factors, different quality properties of software are of importance when designing such systems. Typical examples of system types in Industry 4.0 are systems focusing on smart manufacturing facilities, Internet of Things (IoT) devices, and AI-enabled cyber-physical systems (CPS[2]) such as (semi-)autonomous vehicles. Some of the common quality properties of these systems are real-time constraints, safety, security, power consumption, and certification constraints (compliance with industrial standards often requires following engineering methodologies and processes). While in general information systems these quality properties are often important, in I4.0 these quality properties are the main system driving factors. If some of these requirements is not met, then system can be considered a failure. In case of safety-critical systems, a failure to ensure safety levels can lead to loss of human life or huge financial damage. To ensure that system qualities will be met with solutions, engineering of these I4.0 systems often requires significant effort. This is especially the case due to heavy coupling between software and hardware (effectively characterising them as embedded systems). Introduction of AI into such engineering could potentially make the engineering effort less demanding and help engineers to create systems that will solve more complex problems, while at the same time ensuring that these systems optimally use available computing resources and minimize power consumption.

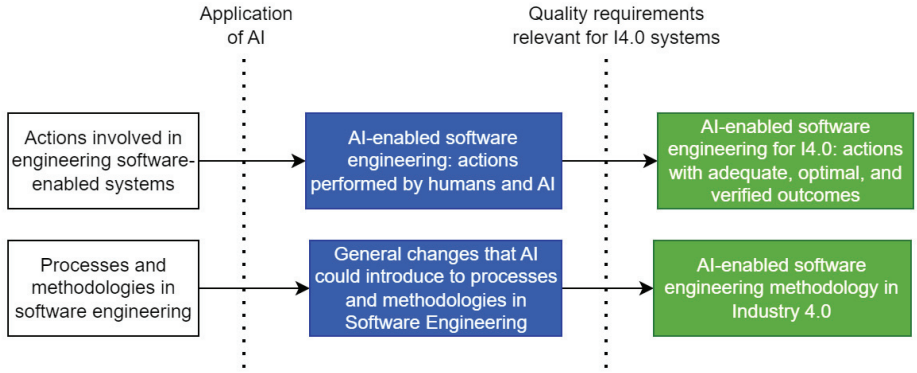

Figure 2: Research methodology

Therefore, in this paper we seek to answer the following question: *How will AI change actions in software engineering performed by humans, and in particular how will that change affect existing software engineering methodologies and processes for software-enabled systems in Industry 4.0?* Methodologies and processes are an important part of software engineering mainly responsible for

[2] https://ecssria.eu/2024_1.4

ensuring that software meets its quality properties. Changes in them will have a profound impact on the way we design, implement, test, deploy, monitor, fix, and extend software solutions. To make predictions on how introduction of AI in software engineering could change its processes and methodologies, we first discuss general changes that AI could introduce in these areas (Figure 2). Then, we gather quality requirements relevant for I4.0 systems and map them to processes and methodologies, as well as software engineering actions, responsible for creating solutions. The outcome of this research is one of the first attempts to predict how AI will change software engineering in I4.0. We also summarise a list of open questions that future research in this domain should answer. This paper introduces two main contributions:

1. Predictions on how AI will change actions, processes, and methodologies in general software engineering.
2. Predictions and discussion on how changes in general software engineering due to adoption of AI will change engineering of I4.0 systems.

This paper is organised as follows. In Section 2, we introduce related work in this domain. In Section 3, we present an abstract view of existing methodologies and processes in software engineering. In Section 4, we present our view on the changes that AI will potentially introduce to engineering of software systems in I4.0. We conclude this paper in Section 5, with a list of open questions for future research.

2. Related Work

Waterfall model [2] was one of the first methodologies for software development. It largely tried to imitate manufacturing processes from factories with focus on creating and following a plan. However, even when the Waterfall was introduced, its authored expressed scepticism regarding some of its parts. The biggest among the issues with Waterfall were long delivery cycles (time from requesting a change in software and delivering a new software version containing that change) and rigidity in following a plan instead the ability to respond to a change request. Agile[3] manifesto tried to put the focus on fast delivery of incremental software updates.

While this has indeed ensured that the created software has desired functionality, lack of long-term plan (roadmap)in Agile often leads to naturally grown architectures. In such architectures, it is very hard to ensure system quality properties such as safety and security, as the focus is on short-term increments instead of long-term system behaviour. Continuous engineering (CE) [3] put the focus on creating a link between business strategy, development, and

[3] https://agilemanifesto.org/

engineering operations. The goal of CE is a constant deployment and delivery of new software versions that are well tested (some companies are capable of delivering and deploying several software versions in the same day). To add actions in this process related to system-level planning (software system architecture), researchers and practitioners introduced continuous architecture [4]. This approach combines continuous engineering with dedicated time and effort for system design and long-term planning, with these actions runnin in parallel.

Industry 4.0 systems often have strong quality requirements (e.g., safety, security, power consumption). There have been several attempts to adjust Agile to work for engineering I4.0 systems and support their quality properties[5][6][7].Besides these, one approach to software development in I4.0 that has gained a lot of popularity in recent years is model driven development. Model driven development focuses on abstraction of component logical actions from implementation, and generation of source code from high level models[8]. It enables planning on a higher abstraction level and configuration of functionality through predefined sets of parameters.

When it comes to adoption of artificial intelligence in software engineering, certain tools are making great breakthroughs (e.g., CoPilot[4]).Scientific research also demonstratesbenefits of AI in software engineering for writing source code and for performing associated actions (mostly by using large language models, such as ChatGPT)[9][10][11]. Other works focus on how AI can assist software architects in creating system design [12]. Despite benefits that the existing approaches demonstrate, they still suffer from low accuracy and low reliability [13][14][15]. There are a quite few attempts to discuss ethics around using AI solutions for engineering of software system. One of the most fundamental works in this domain is the Copenhagen manifesto [16]. It revolves around an idea that AI in software engineering must be human-centred and lists 12 principles to guide adoption of AI in software engineering.
To the best of our knowledge, this work is a pioneering work in both trying to predict i) how AI will change actions, processes, and methodologies in general software engineering, and ii) how changes in general software engineering due to adoption of AI will change engineering of I4.0 systems. While other approaches work on individual challenges, we take the wider software engineering scope into the account and aim to set a track for the whole discipline of software engineering.

[4] https://github.com/features/copilot

3. Common Actions, Methodologies and Processis in Software Engineering

Engineering software systems comprises several actions (Figure 3). Different stakeholders (e.g., customers, business, marketing) have their wishes and concerns (a stakeholder is anyone affected by or concerned with an outcome of a project). Software architects consider wishes and concerns from stakeholders to understand the scope of the system and use them to define drivers that capture system´s problem domain.
In the solution domain, architects seek to make decisions that will address problem domain (system requirements and drivers) and create conceptual solutions that developers need to implement. To do so, architects rely on their own knowledge (experience) and their creativity. If we assume that there is a space that contains all adequate solutions for drivers of a certain system, then only a portion of those is known to architects or documented in general. The challenge for architects is how to reuse solutions unknown to them, but otherwise documented. Conceptual solutions created by architects serve as a blueprint for software implementation. It is necessary to support implementation effort with operations, such as building binaries, testing them, integrating contributions from different teams into a software solution, deployment of different software versions to target environment and their delivery to end users. As part of system execution, it might be also necessary to monitor the system and take adequate actions based on the monitoring results.

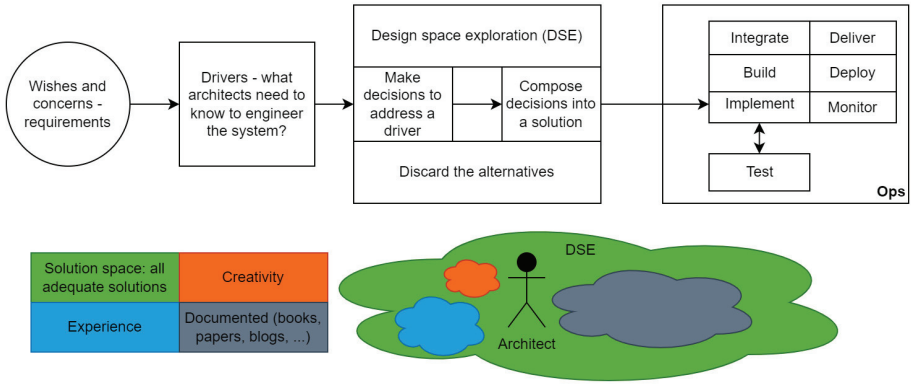

Figure 3: Actions involved in engineering software-enabled systems

Actions discussed above exist in every software project to a greater or a lesser degree. Engineers of software systems can organise their work (mentioned

actions)using different procedures and schedules (e.g., Waterfall[2], Agile[5]). Let us discuss one example of such organisation based on combination of Agile Scrum[6](Figure 4) and continuous architecture [4].

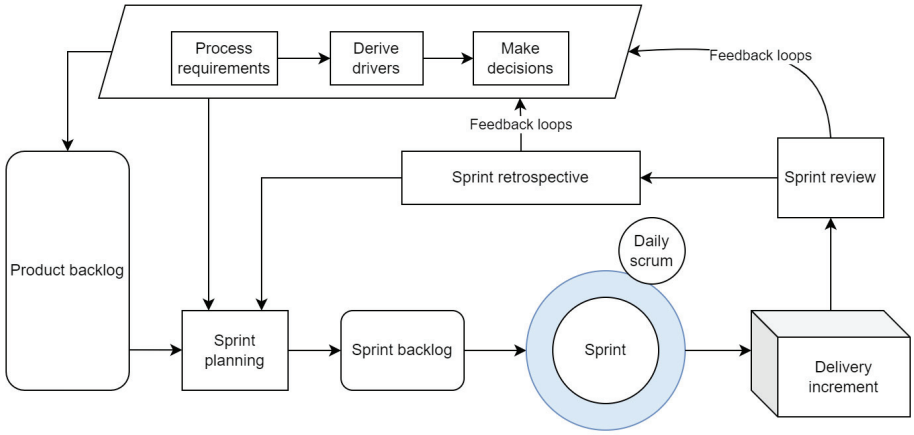

Figure 4: An example of organisation of continuous architecture using Scrum development methodology

In this example of the work organisation (processes and methodology), architects process requirements, derive system drivers, and based on these make architectural decisions about the system. The outcome of these activities is a (updated) project roadmap. Engineers of a system use the roadmap to a create/update a prioritized list of work (***product backlog***). In Scrum, engineering activities are organised in sprints. A sprint usually takes two to four weeks. During ***sprint planning***, engineers prioritise activities from the backlog that can reasonable be implemented within the duration of one sprint (***sprint backlog***). Each day, engineers have short standup meetings (***daily scrum***), during which they provide brief update on their work. At the end of the sprint, engineers deliver an increment to the previous product (***delivery increment***). During ***sprint review***, the team presents to stakeholders the delivery increment and gathers feedback from them.Internally, the team working on the system performs ***sprint retrospective*** to determine what went well during the sprint and what could be improved. Outputs from the sprint review and sprint retrospective are fed to the system architects who then decide about the roadmap.

It is important to notice here that actions specified in Figure 3 are present in engineering every software-enabled system. They can be explicitly specified and

[5] https://agilemanifesto.org/

[6] https://www.scrum.org/resources/what-scrum-module

performed or handled implicitly. Engineers working on software systems find a way to organise their work. That organisation can conform to one of the existing development methodologies (e.g., Waterfall) or can be customised. Regardless, there is always either explicit or implicit organisation of work.

4. Influence of AI on Methodologies and Processes in Software Engineering

In the previous section (Section 3), we have discussed common actions, methodologies, and processes in software engineering. Let us first discuss how artificial intelligence is changing these (Section 4.1). To understand how is this relevant for I4.0 systems in particular, we discuss quality properties of these systems derived from different relevant industry standards (Section 4.2). Finally, we try to predict how AI will change actions in software engineering performed by humans and consequently processes around such software engineering, with the focus on systems in I4.0.

4.1. AI and Software Engineering

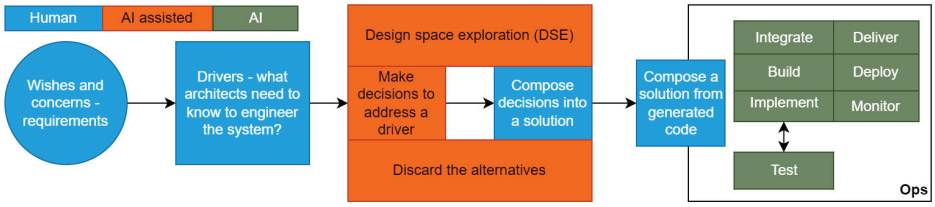

Figure 5: AI-enabled software engineering: coverage of work by humans and AI

In the last few years, we have witnessed significant developments in artificial intelligence and its application in software engineering (as discussed in Section 2). This development will have a significant impact on actions that humans perform in (future) software engineering.The key questionswe are starting to ask is, which of the software engineering actions will be performed by humans, which of these actions will be supported by AI, which actions are suitable to be fully replaced by AI, and what are potentially new actions that engineers will perform. In other words, if AI is able to replace engineers doing some actions, what is it that humans bring to the table? To answer this question, we observe influence that introductionof AI has in the context of software engineering (Figure 5).

Understanding the context of a problem that a system should solve still remains in the domain of human intelligence. Even the most advance AI models today face the challenge of comprehending the problem domain. One example that

illustrates this is the case of stopping Waymo's robotaxis after they encountered a man with a t-shirt that contains STOP sign[7]. Another example is the AI chatbot provided by Google that advised people to eat one stone per day as stones might contain minerals[8]. Previous research also confirms these conclusions regarding AI and its struggle with the contextual understanding of problems it needs to solve. Experiments with application of AI in software system architecture have demonstrated that AI models are unable to fully comprehend context for which they are generating solutions[12]. These experiments show that AI cannot differentiate between different abstraction levels – effectively meaning that it cannot reasonwhat is more important to architects who consume such AI generated solutions. If AI will ever be able to replace a human or be of an assistance to humans in this domain, then such AI would be (very close to) artificial general intelligence (AGI)[17] and would have expert domain knowledge about domain in which we try to apply it. Furthermore, as AI cannot grasp the context of stakeholders, their background, wishes and concerns, it also cannot differentiate between what is it that indeed drives software projects (we simply cannot capture all implicit requirements, it still takes a human to know). Such AI would need to have knowledge of advanced social skills and broad civic and domain knowledge. Consequently, it means that AI is unsuitable for deriving drivers. In conclusion, we can say that for defining a problem domain (motivation for creating a system, reasons for it to exist, domain in which it should exist with associated limitations and assumptions), AI at the moment (and in foreseeable future) is not of much value.

Architectural decisions are first artefacts in the solution space that engineers produce when developingsoftware-enabled systems. Previous research [12] has demonstrated that AI can assist with interpreting knowledge that exists in software architecture domain and assist in this process. This research shows that AI is also suitable for derivingnew solution patterns previously not considered by architects. Therefore, for design space exploration, already today AI can act as an assistance tool, as it can suggest solutions for individual (smaller scale) architecture problems. However, AI does not have a notion of right and wrong (correct and incorrect, true or false). Furthermore, it is still struggling to compose solutions that would be adequate for a combination of explicitly expressed and assumed systems´ requirements composing alarger problem domain. Therefore, composing individual system architecture decisions into a system solution remains in the hands of human engineers and architects.

[7] https://www.carexpert.com.au/car-news/how-a-t-shirt-stopped-this-autonomous-car-in-its-tracks

[8] https://www.cnet.com/tech/services-and-software/glue-in-pizza-eat-rocks-googles-ai-search-is-mocked-for-bizarre-answers/

When it comes to code generation, test generation, and supporting other operations in software engineering, AI proved to perform quite well[18][11]. For now, AI still performs these activities on a small scale. But as it has been shown to be the case in other engineering areas, there is no doubt that AI is capable of generating solutions that outperform human engineers due to its ability to extract complex patterns from data or connect huge number of data points and data dependencies not obvious to humans when creating a solution[9]. Therefore, we can expect that in future AI can take over writing the code for individual problems, generating tests, interpreting the code to new engineers, writing scripts for deployment and delivery, and maintaining the whole operations infrastructure. However, what still remains to be a challenge is generation of full solutions for dedicated design. Much of the design that architects provide is abstracted. In such context, developers have implicit understanding with architects how to implement systems to conform to the prescribed architectural design. Eliminating humans from the loop would mean eliminating this implicit understanding. Therefore, it is realistic to expect that humans will still perform the role of an integrator of code components individually generated by AI.

4.2. Key Software Quality Properties in Industry 4.0

Industry 4.0 covers a large group of system types. These are buildaround artificial intelligence, cyber-physical systems, Internet of Things (IoT), and cloud computing. These underlying technologies enable the most important properties of I4.0, such as connectivity, distributed computing, and human-centric engineering. Let us observe key quality architectural properties of these systems and let us discussstandards that prescribe their engineering.

In terms of engineering, adopting artificial intelligence these daysis mainly associated with two sets of actions: training and inference. While programming and using systems based on trained AI models is not a particularly challenging task, what remains as a big issue is the ***power consumption***when running these models. The computing power required for AI is doubling every 100 days [19]. It is projected that this requirement will increase by more than a million times over the next 5 years [19]. Besides power consumption, another challenge is what does the adoption of AI mean for certification of ***safety***-critical products (e.g., autonomous vehicles). Besides challenges with energy consumption, second greatest challenge with AI is the inability to properly test AI-based systems (especially significant constraint for safety-critical systems).

In cyber-physical systems (CPS), challenges are present around ***safety*** and ***security*** in the first place. Furthermore, there are also challenges related to management of such systems (***management of complexity, maintaining legacy***

[9] https://spectrum.ieee.org/ai-3d-printing-better-ac

systems, updating and extendingthe systems). When CPS are a bases for other systems, such is autonomous vehicles or avionics, there are strict standards that prescribe engineering processes and methodologies. Only by following these standards, it is possible to receive necessary ***certification*** for putting these systems on market. One of the most distinct quality properties in these systems are ***real-time*** constraints.

Systems composing Internet of Things (IoT) are dominated by low-power sensors and low-power communication and control devices. Therefore, in these systems ***power consumption*** and ***securecommunication*** are the most important quality drivers. Besides these quality properties, ***reliability*** of such systems is important as well as the ability to ***dynamically extend*** them with adding new sensor and computing nodes.

Cloud-based systems are mostly concerned with ***optimal use of available computing hardware*** and with ***power consumption***, where these two properties are tightly coupled. For example, finding an optimal scheduling of workloads for available computing nodes greatly saves power consumption, making the whole system more energy efficient.

Here is a summary ofthe most relevant standards that exist in these domains:

- IS26262[10] is a safety standard for road vehicles. It prescribes development of safety-critical products on system, hardware, and software level.
- IEC 61508[11] is an international standard for Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (ISO 26262 is an adaptation of this standard). It prescribes different safety integrity levels, and according to each of these levels, it prescribes types of required testing (e.g., functional coverage for the lowest safety level, MC/DC coverage for the highest safety level).
- ISO 21448[12] focuses on safety of the intended functionality. For that, it provides guidance on the applicable design, verification and validation measures.
- IEEE 2851[13]is a standard for Functional Safety Data Format for Interoperability within the Dependability Lifecycle. It describes methods, description languages, data models, and database schema to enable the exchange/interoperability of data across all steps of the product´s lifecycle.
- ISO/SAE 21434[14] is a cybersecurity engineering standard for concept, product development, production, operation, maintenance and

[10] https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en
[11] https://www.tuvsud.com/en-us/services/functional-safety/iec-61508
[12] https://www.iso.org/standard/77490.html
[13] https://standards.ieee.org/ieee/2851/10780/
[14] https://www.iso.org/standard/70918.html

decommissioning of electrical and electronic (E/E) systems in road vehicles, including their components and interfaces.
- ISO/DPAS 8800[15] standard prescribes safety-related properties and risk factors impacting the insufficient performance and malfunctioning behaviour of Artificial Intelligence (AI) within a road vehicle context.
- ISO/IEC TR 5469[16] standard describes the properties, related risk factors, available methods and processes related to i) use of AI inside a safety related function to realize its functionality, ii) use of non-AI safety related functions to ensure safety for an AI controlled equipment, and iii) use of AI systems to design and develop safety related functions.
- DO-178C[17] standard focuses on software in airborne systems and their equipment. It focuses on tools, model-based development, verification, formal methods, and safety considerations.

From the summary of technologies commonly present in I4.0 and the summary of the relevant standards, we can conclude that the key quality properties relevant for I4.0 systems are:
- power consumption
- optimal use of available computing hardware
- safety
- security
- management of complexity, maintenance of legacy systems, updates and (dynamic) system extensions
- real-time
- reliability
- certification

At the same time, while meeting these requirements, systems also must be performant to exercise its functionalities, cost-efficient, and profitable.

4.3. AI-enabled Software Engineering in Industry 4.0

In Section 4.1, we have summarised actions, methodologies, and processes in software engineering that AI could take over, could assist with, and those that require exclusive human involvement. In the previous section (Section 4.2), we have summarised the most important quality properties of I4.0 systems. Let us merge these two and discuss how AI can potentially change software engineering for I4.0.Outcomes of actions in software engineering can be

[15] https://www.iso.org/standard/83303.html
[16] https://www.iso.org/standard/81283.html
[17] https://www.rtca.org/training/do-178c-training/

adequate (good enough), optimal, and formally verified.AI does not provide guarantees about formal verification of its generated solutions. However, could help to guide formal verification of solutions. Furthermore, considering large number of parameters that condition trade-offs between system quality parameters, AI can provide optimal solutions that humans cannot even comprehend.

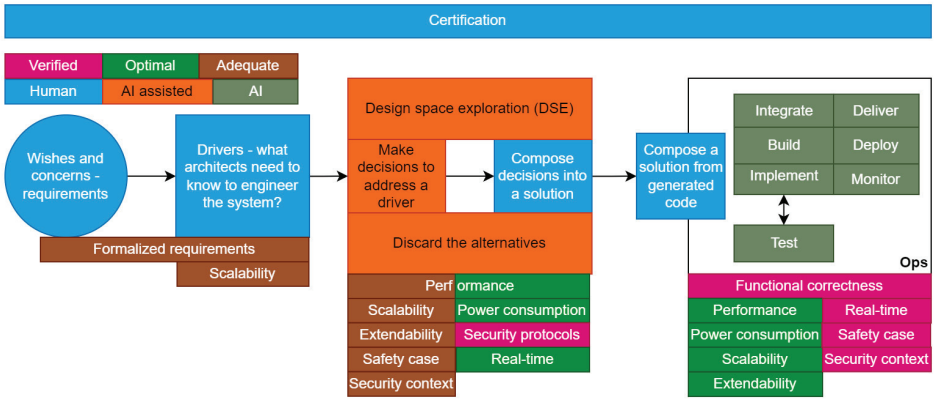

Figure 6: Different levels of quality of solutions: adequate, optimal, verified, and how AI can help to support them

As previously stated, understanding the problem domain is a task for humans. AI hardly can help in this domain.

On the other hand, creating architectural solutions is an action where AI can assist with architectural decisions. While providing optimally performant design for the whole system is hard to generate (especially considering that AI cannot even grasp the context in which the system should operate and architects cannot provide all possible details), providing optimal solutions for individual problems is possible. In that context, AI could help to generate optimal solutions such is scheduling and deployment configurations, considering different workloads, communication latencies, network configurations, and available computing nodes. Considering that architecture is an abstraction, it does not contain all the details that could facilitate optimal design performed by AI. In that context, it is also important to emphasize that systems evolve and do so in a way that is hard to predict. While AI can assist with design decisions, considering quality properties such as scalability, extendability, and building safety and security cases it is only possible to create adequate designs. On the other hand, considering concrete problems such as power consumption in a design context with enough information and real-time constraints (e.g., scheduling policies, priorities, available computing nodes), AI has potential to generate optimal design solutions.It is important here to emphasize once more the ability of AI to

discover dependencies between quality properties normally not obvious to humans. Therefore, besides other benefits, AI could be used to suggest optimal trade-offs between key quality properties. However, AI remains inadequate for verification of concrete design solutions, such is verification of security protocols. Verification of these still requires formal methods.

On the implementation level, however, AI has a lot more potential. We have created high-level programming languages to enable easier writing of code, management of large code bases, and management of complexity that stems from large code bases. However, there is no obstacle that prevents AI from generating solutions directly in assembly code, optimised for the target instruction set architectures (ISA). In that context, AI has a potential to generate optimised implementations in form of binaries (it still remains an open question how engineers would handle this, if AI is not able to generate full solutions but pieces of the solution that engineers have to integrate). In that context, AI could optimise binaries also for power consumption. Furthermore, generated solutions could take into the account target deployment platform. AI, with its ability to understand far more dependencies in software system solutions than humans, could optimally extend existing solutions with new features directly in the source code. Although AI cannot guarantee formal verification, we could expect that it generates functionally correct code, eliminating the need for testing on the lower levels (e.g., unit test level). This puts validation of the solution in the perspective, where AI testing could act as fuzzy testing[20] on higher abstraction levels (e.g., integration and system testing).Lifting the testing and validation into a more abstract domain definitely has benefits, as long as humans can keep some level of the validation and testing control. If AI is able to observe the functional rules of programming languages, we could claim that it (almost) achieves formally verified functionally correct source code. While AI cannot formally verify real-time constraints, safety case, and security in a broader context, it can help to generate solutions that are optimised towards these goals.

The major question, however, still remains. What does this mean for certification of such systems? At this point, we do not have a clear answer to this question, and it remains an open point for future work.

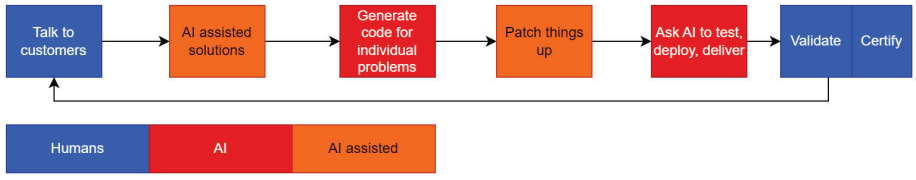

Figure 7: A prototype of an AI-enabled software engineering methodology

In Section 3 (Figure 4), we have introduced an example of methodology that supports continuous software architecture. Considering the potential benefits that AI could offer to software engineers and architects, the question is how that influences software engineering methodologies (Figure 7)? Does it deem certain actions unnecessary and creates a need for others? Interaction with those that define the problem domain still is in the domain of human work. In the solution domain, engineers can use AI tools to reach adequate (and in some cases optimal) solutions. If we are truly able to generate code snippets for well-defined problems, then the job of an engineer is to understand such solutions (to make sure that they do not do something different from the expressed requirements), and patch them into an integrated solution. Under the assumption that the generated code is functionally correct, engineers can use AI to test the solutions on the higher abstraction level and ask AI to generate scripts that will handle other operations (e.g., deploy, deliver). However, it is still the task of the humans to validate if the produced solutions satisfy requirements. If we cannot rely on AI to capture requirements and context around them, then we cannot rely on AI to validate solutions (impossible to validate if AI does not understand against what it needs to validate the solution).
Consequently, it is also the job of humans to deal with the certification of safety-critical products. There is no doubt that certification in future will change. It is quite likely that there will be a possibility to engage into a process of continuous certification through continuous compliance with standards [21], reducing the effort and time for verifying evolving safety-critical systems. However, it is also clear that humans have to remain the essential part of that process (because only humans understand the context in which solutions will perform).

One model that illustrates the discussion above in a quite nice way is (somewhat archaic but still relevant) V-model[22]. The V-model is a development process applied in many domains, including software system engineering. The main idea of the V-model is verification and validation of all actions in software projects. Validation is the assurance that a system meets acceptance and suitability criteria made by (external) stakeholders and their high-level requirements. Verification is the assurance that a system meets a defined set of design specifications, shows absence of bugs and meets quality specifications (for example, including standards).
The main elements of the V-model are:

- Concept of operations: describes stakeholder needs and the operating environment. This action considers writing the list of stakeholders and their operations in the system.
- Requirements and architecture: suggests collecting detailed requirements about how an ideal system should perform by analysing stakeholder needs

and suggesting techniques and approaches for meeting these needs. This action should also list requirements that are not feasible.

- Detailed design: the architectural design in this stage is detailed further so it is ready for implementation.

After implementation, the project undergoes testing and integration activities:

- System verification and validation. System tests are set up by a business team to test the overall functionality and quality of a system.
- Integration, test and verification. Integration tests verify if parts of the system that are developed independently can communicate, collaborate, and coexist among themselves. On the lowest abstraction level are unit tests. These tests verify if individual system parts can function correctly, isolated from the rest of the system.
- Operations and maintenance. Once tested, the project enters the operation and maintenance stage.

Although Waterfall-like structure behind V-model might not be suitable for the majority of project these days, it still captures essential actions and dependencies between them that each software engineering project should perform. When we apply the conclusions from this chapter on how AI is changing software engineering, we get division of work in V-model as illustrated in Figure 8.

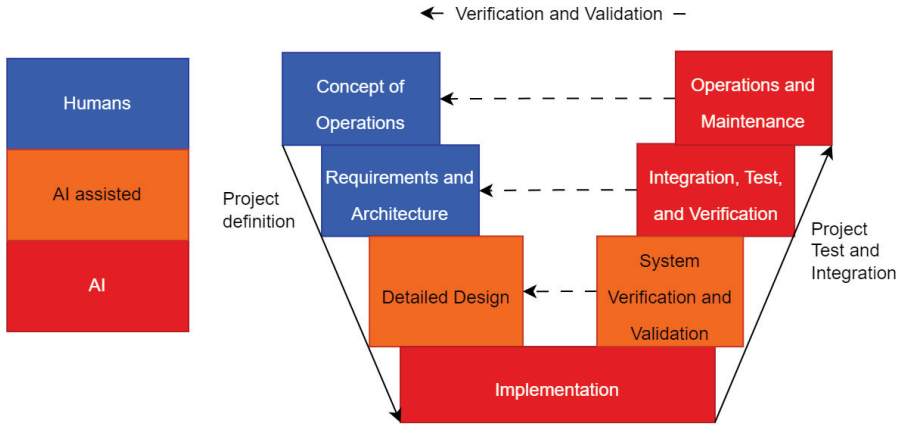

Figure 8: V-model complemented with AI-enabled software engineering: coverage of work by humans and AI

5. Conclusion

Adoption of AI in software engineering is current state of practice. This adoption will only increase in future. However, researchers and practitioners struggle to adopt AI in their engineering processes and methodologies in a way that would maximise their benefits. To deal with this problem, we have observed what kind of actions AI can take over from humans and where it can assist. Furthermore, we also discussed in this paper why some of the actions will most probably remain under the responsibility of humans. The conclusion is that adoption of AI in software engineering might completely re-shape this domain, in terms of how we organise work, responsibilities of developers, perceived value of delivery, and focus of invested effort in projects. For example, it might happen that writing code, tests, and scripts for all associated operations might represent only a small fraction of software costs, where bringing in other experts (such as experts on performance, distributed systems, UI, safety, security, etc.) might be the focus when engineering future systems.

In this paper, we have derived new sets of actions that engineers might be performing in AI-enabled software engineering for I4.0. We have suggested how to organise those actions into a new methodology. Finally, here we list a set of open questions that we consider that future research in this domain must try to answer:

- If AI increases productivity of software engineers to the point where a single engineer can deliver the same or more as previously a team of engineers, will we still need software development methodologies such as Agile? In that case, will the software engineering task be reduced in effort (to only connecting components generated by AI) and will we dedicate more resources to domain experts (performance architects, safety and security experts, UI experts, etc.)?
- If AI is able to generate correct code, will there still be a need for low level (e.g., unit level) testing?
- If AI can generate code for small problems, will the focus of engineers move to integrating small generated solutions to solve big problems?
- If AI can understand machine code, is there a need anymore for higher level programming languages? Can AI instead generate machine code from high level specifications?
- If AI can understand machine code, can it edit, update, and extend software on the machine code level directly?
- If AI can easily rewrite (machine code) solutions, does code have the same value anymore as (we think) it has today? Is there a need, in that case, for any documentation concerning source code itself?
- Can AI assist with guiding tools and processes for formal verification?

6. References

[1] B. Chunguang, P. Dallasega, G. Orzes and J. Sarkis, “Industry 4.0 technologies assessment: A sustainability perspective,” *International Journal of Production Economics,* pp. Volume 229,, 2020.

[2] W. Royce, «Managing the development of large software systems: concepts and techniques,» chez *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, 1987.

[3] B. Fitzgerald et K.-J. Stol, «Continuous software engineering: A roadmap and agenda,» *Journal of Systems and Software,* vol. 123, pp. 176-189, 2017.

[4] M. Erder et P. Pureur, Continuous Architecture - Sustainable Architecture in an Agile and Cloud-Centric World, Morgan Kaufmann, 2016.

[5] R. Dovleac, A. Ionica et M. Leba, «Knowledge Management Embedded in Agile Methodology for Quality 4.0,» chez *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Singapore, Singapore, 2021.

[6] H. B. Christensen, S. C. Jepsen et T. Worm, «Agile Architecting of Distributed Systems for Flexible Industry 4.0,» chez *16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, Sofia, Bulgaria, 2021.

[7] A. Marnewick et C. Marnewick, «The Ability of Project Managers to Implement Industry 4.0-Related Projects,» *IEEE Access,* vol. 8, pp. 314-324, 2020.

[8] D. Schmidt, «Model-Driven Engineering,» *IEEE Computer,* vol. 39, n° %12, 2006.

[9] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin et X. Hu, «Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond,» *ACM Transactions on Knowledge Discovery from Data,* vol. 18, n° %16, 2024.

[10] Z. Zhang, C. Chen, B. Liu, C. Liao, Z. :. Y. H. Gong, J. Li et R. Wang, «Unifying the Perspectives of NLP and Software Engineering: A Survey on Language Models for Code,» 10.13140/RG.2.2.32710.69440/1, 2024.

[11] Z. Z. Chen, K. Ning, Y. Wang, J. Zhang, D. Zheng, M. Ye et Jiachi, «A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends,» https://arxiv.org/abs/2311.10372, 2024.

[12] J. Jahic et A. Sami, «State of Practice: LLMs in Software Engineering and Software Architecture,» chez *21st IEEE International Conference on Software Architecture (ICSA 2024)*, Hyderabad, India, 2024.

[13] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White et D. Poshyvanyk, «An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation,» chez *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Montpellier, France, 2018.

[14] M. Ciniselli, N. Cooper, L. Pascarella, D. Poshyvanyk, M. Di Penta et G. Bavota, «An Empirical Study on the Usage of {BERT} Models for Code Completion,» chez *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*, Madrid, Spain, 2021.

[15] A. Mastropaolo, S. Scalabrino, N. Cooper, D. Nader-Palacio, D. Poshyvanyk, R. Oliveto et G. Bavota, «Studying the Usage of Text-To-Text Transfer Transformer to Support Code-Related Tasks,» chez *18th International Conference on Mining Software Repositories*, 2021.

[16] D. Russo, S. Baltes, N. van Berkel, P. Avgeriou, F. Calefato, B. Cabrero-Daniel, G. Catolino, J. Cito, N. Ernst, T. Fritz, H. Hata, R. Holmes, M. Izadi et F. Khomh, «Journal of Systems and Software,» *Generative AI in Software Engineering Must Be Human-Centered: The Copenhagen Manifesto,* p. 10.1016/j.jss.2024.112115, May 2024.

[17] S. Baum, «A Survey of Artificial General Intelligence Projects for Ethics, Risk, and Policy,» *SSRN Electronic Journal,* vol. 10.2139/ssrn.3070741, 2017.

[18] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang et X. Xie, «A Survey on Evaluation of Large Language Model,» *ACM Trans. Intell. Syst. Technol.,* vol. 15, n° %139, p. 45, June 2024.

[19] S. Zhu, T. Yu, T. Xu, H. Chen, S. Dustdar, S. Gigan, D. Gunduz, E. Hossain, Y. Jin, F. Lin, B. Liu, Z. Wan, J. Zhang, Z. Zhao, W. Zhu, Z. Chen, T. S. Durrani, H. Wang, J. Wu et T. Zh, «Intelligent Computing: The Latest Advances, Challenges, and Future,» *Intelligent Computing,* vol. 2, n° %16, 2023.

[20] A. Nappa et E. Blázquez, Fuzzing Against the Machine: Automate Vulnerability Research with Emulated IoT Devices on Qemu, Packt Publishing, Limited, 2023 .

[21] T. Santilli, «Characterizing Software Architectural Metrics for Continuous Compliance in the Automotive Domain,» chez *21st IEEE International Conference on Software Architecture (ICSA 2024)*, Hyderabad, India, 2024.

[22] K. Forsberg et H. Mooz, «The Relationship of System Engineering to the Project Cycle,» *Incose,* vol. 1, n° %11, pp. https://doi.org/10.1002/j.2334-5837.1991.tb01484.x, 1991.