Generative AI for Software Practitioners

Article *in* IEEE Software · July 2023

DOI: 10.1109/MS.2023.3265877

2 authors, including:

Christof Ebert
Vector Consulting Services
345 PUBLICATIONS **6,627** CITATIONS

Accepted paper. Cite as:
Ebert, C. and P.Louridas: "*Generative AI for Software Practitioners*". IEEE Software, Vol. 40, No. 4, pp. 30-38, Jul/Aug. 2023.
https://doi.org/10.1109/MS.2023.3265877

Generative AI for Software Practitioners

Christof Ebert, Vector Consulting Services
Panos Louridas, Athens University of Economics and Business

Generative AI tools such as Bard, ChatGPT and CoPilot have rapidly gained widespread usage. They also have the potential to boost software engineering productivity. In this article we elaborate technologies and usage of Generative AI in the software industry. We address questions such as: How does Generative Ai improve software productivity? How to connect Generative AI to software development, and what are the risks? Which technologies have what sort of benefits? Practitioner guidance and case studies are shared from our industry context. I look forward to hearing from you about this column and the technologies that matter most for your work.

—Christof Ebert

DOI Keywords: Generative AI, ChatGPT, productivity, software engineering, software technology

Generative AI has the potential to change the software profession more than any other recent technology. Bill Gates sees it as the biggest move forward since the invention of the Internet. It can improve software productivity in several ways, such as automating repetitive tasks such as testing or requirements traceability, improve software quality by creating test suites from requirements, and automating workflows by routing work products to the next suitable step in a production pipeline. At the same time generative AI creates fully new risks because it is neither deterministic nor explainable. IPR and Cybersecurity are prominent examples that limit usage in professional software engineering.

Generative AI Technologies

Generative AI has been around for many years. With no means to prove validity, researchers hesitated to bring such technology to the mass market of rather naïve data citizens. As we have observed many times in recent IT history, the perceived gold rush makes people closing eyes in front of obvious risks. Even tools designed for good will eventually have devastating consequences. When ChatGPT was finally released to a wide public audience in 2022, the AI arms race started at a speed never seen before. It took for ChatGPT just two months to reach a hundred million users. Fig. 1 shows this fast evolution for different technologies spanning a mere hundred years of recent human history. A technology like the wheel even took thousands of years to reach a hundred million users.

For every developer, turning to StackOverflow or Google has been a natural part of the job for years now. Condemning the Not Invented Here Syndrome to the dustbin, our first reaction when in doubt how to code something has been to look it up on the internet. Search engines have got better at indexing code repositories, myriads of which exist online, and community advice sites, such as StackOverflow, provide reasoned solutions and valuable commentary on user questions. What is common in search engines and question and answer websites is that you can lookup information that has already been stored there.

Generative AI is different. As the name suggests, it can synthesize, or generate, the answers to the questions you pose. Instead of trawling a prefabricated answer as classic search engines are doing, it will create an answer for you. The answer is based on vast amounts of data on which it has been trained, such as those archived and indexed by search engines. To provide meaningful answers, Generative AI undergoes further training based on human feedback. Many human trainers pose questions and provide feedback on the generated answers, rewarding good answers and punishing unsatisfying ones. This kind of reinforcement learning guides the system towards providing more accurate answers, while guarding against harmful responses. This has led to glimpses of a new way of working, where the focus is on "prompt engineering": find the most appropriate way to frame a question, or a whole dialog. Generative AI does not work with individual question and answers: it maintains a context window, which can be used to guide the AI in generating contextually relevant and well-informed responses.

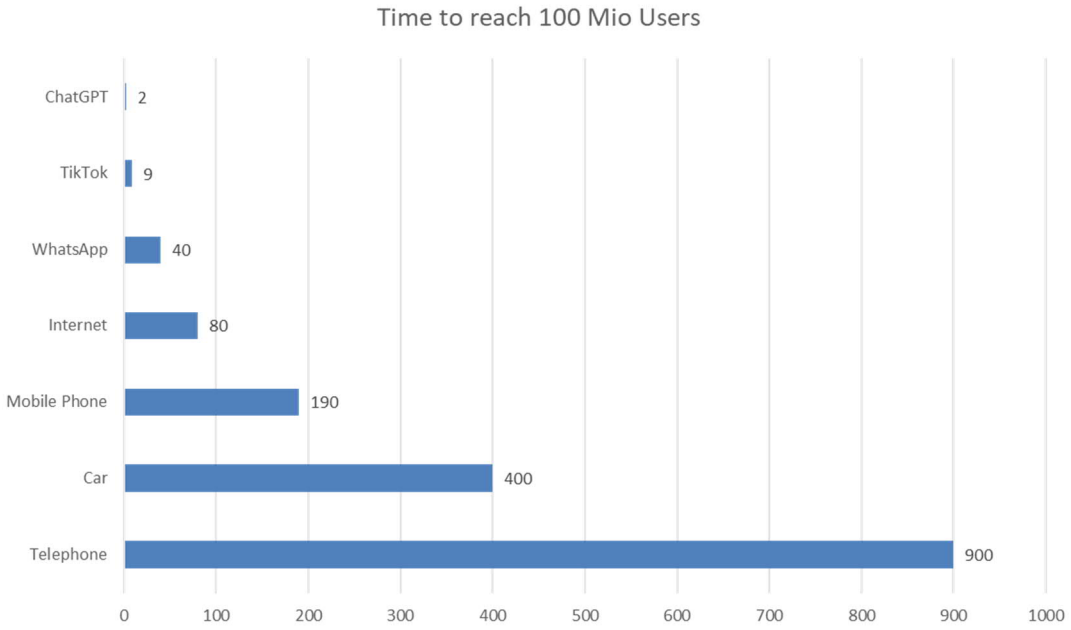

Fig. 1: Time to reach 100 million users for different technologies in months after initial deployment.

Underlying all this, Generative AI is power by Large Language Models (LLMs). As the name suggests, these are large neural network models that are trained on big language corpora. Technically, they have a *Transformer* architecture, which is based on a mechanism called *attention*. The publication of the attention mechanism, by Google researchers, must now rank among the most influential papers in Computer Science [1]. Two early LLMs were BERT (Bidirectional Encoder Representations from Transformers), developed by Google in 2018 [2], and GPT-1 (Generative Pre-trained Transformer 1), developed by OpenAI who went on to develop subsequent GPT models, getting to GPT-4 today [3].

The basic idea is to use a large language corpus to train a neural network to learn the language, by hiding part of the text and asking the network to guess the missing parts [4]. The neural network does that by paying selective attention to the words comprising the surrounding context of the missing parts. The words themselves are represented as vectors, called *embeddings*, in a multi-dimensional space. In essence, the neural network learns, and represents the meaning of each word in a numerical format, which can then be processed by the AI model. This is achieved by mapping words to vectors in a high-dimensional space, where words with similar meanings are situated closer together. These probabilistic representations can then be used to generate entirely new text – for instance, the answer to a query. When we talk about language and words, we are not restricted to human language: it can be computer code, having code tokens instead of words; the idea is the same.

You can see a simplified depiction of the internals of a LLM in Fig. 2. The model follows the transformer architecture [4]. The inputs, which are language or code tokens, are represented as vector embeddings. Then they go through an *encoder*, which is a series of attention mechanisms. Attention mechanisms are an algorithm used in LLMs that enables the AI to focus on specific parts of the input text when generating an output. The

output of the encoder is a vector representation of the input, which is produced by analyzing surrounding context and attentions. You can think of the encoder's output as the meaning of the input, as understood by the neural network.

Once we have the encoded input, we need to transform it to the desired output, that is, take it from a vector representation and transform it back to a language or code token. To do this we feed it to another series of attention mechanisms, the *decoder*. The output of the decoder are candidate tokens, which are then assigned probabilities. The most probable token is the final output. These probabilities result from training the entire transformer model, including both the encoder and decoder, with vast amounts of text. ChatGPT is said to be trained with the "entire internet". The training process is referred to as "self-supervised learning" (or "masked language modeling"), which is achieved by hiding some parts of known text and checking the quality of how it is automatically completed. In this way the decoder will learn to predict the missing output given the encoded input. After training, the model receives prompts or queries. Each prompt will be encoded as before and then fed to the decoder. This time the decoder works only with the encoded input as we don't have a known output at hand. With adequate training, the model will predict a useful final output. Obviously with restricted domains, such as code fragments or test cases, the LLM needs less training.

Fig. 2 corresponds to the basic model and training procedure; for a more detailed overview of LLMs see [4]. As we mentioned above, a vital part of Generative AI models is that once they can produce sensible outputs from a prompt, they can be further trained by having humans providing feedback to their output. The feedback is used to fine-tune the model using reinforcement learning techniques.

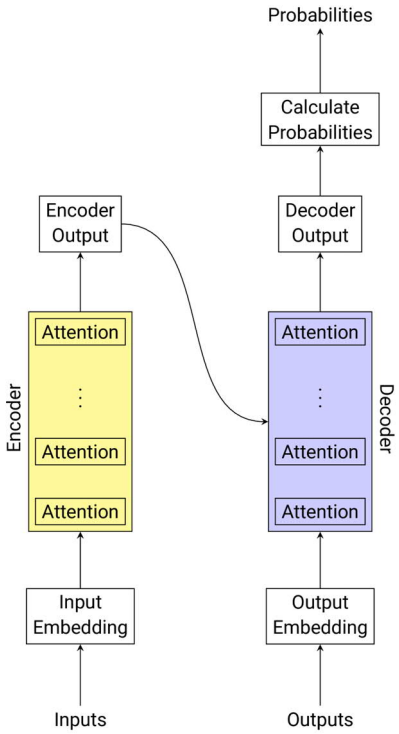

Fig.2: Generative AI technology

Generative AI for Developers

Several generative software platforms are available, which allow to turn simple instructions into computer code. GitHub Copilot is available as an extension to development tools and editors, such as Visual Studio Code, Visual Studio, Neovim and JetBrain IDEs. It offers code autocomplete that is powered by OpenAI Codex, a Generative AI system developed also by OpenAI. A tantalizing development is the recent announcement of Copilot X, powered by GPT-4, which apart from improved autocompletion can aid in other development tasks, such as understanding code, improving pull requests, and aiding in scripting and shell tools.

GPT-4 can generate code from docstrings and solve coding questions in software engineering interviews on a par or surpassing human performance. It can program for the front-end and interact with LaTeX. It can reverse engineer code, execute Python code, and execute pseudo code. OpenAI, the company behind GPT-4, offers programmatic access to its LLMs. That means that developers can use them not only in conversational manner, but also embed them into their applications. It is also possible to develop plugins, which are ways to connect the underlying models with third-party services that can respond to questions and act on them.

Code completion, at a line or whole function level is offered by Tabnine, which positions itself apart from the competition by paying special attention to licensing and privacy issues. Tabnine has been trained only on open-source software with permissive licenses. Moreover, it assures developers that it does not retain any of the code in which it is used, and it is also possible to download and use the underlying models locally, instead of only being accessible as a service.

Table 1 summarizes some mainstream technologies. Note that the landscape changes very fast. If anything, we can expect existing tools to improve, and new tools to enter the scene.

Table 1: Generative AI technologies for Code Development

Name	URL	Technology	Cost	Use cases
ChatGPT	https://chat.openai.com/	GPT-4	$20 per month for the chat interface; pricing for API use depends on usage.	Code completion, code generation, code comprehension, reverse engineering, pseudo code reasoning and execution.
CoPilot	https://github.com/features/copilot	OpenAI CodeX, GPT-4	$10 per month/$100 per year for individuals, $19 per user per month for business plans.	Code completion for CoPilot; CoPilot X uses the more advanced GPT-4 model and can answer questions based on code documentation; aid in pull requests, shell commands and scripting.
Tabnine	https://www.tabnine.com/	Proprietary ML engine trained with OSS	Basic teer is free, Pro teer starts at $12 month / user, also possible to self-host	Code completion. Runs also on private desktop to protect IPR.
Hugging Face (various different models)	https://huggingface.co/docs/transformers/index	Transformer, details vary depending on the model	Free and Open Source	Code completion and code generation, depending on the model.

Generative AI for Software Productivity

Generative AI has the potential to significantly improve software productivity by automating tasks, enhancing creativity, improving accuracy and efficiency, and streamlining development processes. Application domains with a high business impact are the following:

- Media content creation, e.g., text, audio, video, pictures, content for news feed and social media
- Media content improvement, i.e., from making enhancements, references (though mostly they hallucinate and invent references that do not exist), and explanations up to augmented content generation with pay per use.
- Generative design, such as chip-design in semiconductor business, developing novel building and city architectures, material design in chemical plants, innovative drugs and medication in pharma industries.

- Software development, such as code generation, test case generation from requirements, re-establishing traceability, explaining code, refactoring of legacy code, software maintenance with augmented guidance and improving existing code.

We will focus here on improving software development and software productivity with generative AI (see sidebar with “Generative AI Industry Case Study” for an application).

Even though we have focused on code-related tasks, the capabilities of generative AI tools extend well beyond assisting in writing code. Generative AI can improve software productivity in various aspects of Software Engineering, such as:

Enhancing creativity: Generative AI can assist developers in generating new ideas and solutions for software und UX development. For example, it can help developers to generate new designs, logos, and user interfaces.

Summarizing documentation, reviews, interviews, meeting minutes: LLMs are particularly good at these tasks, freeing time for other tasks. Generative AI responds to prompts, and these prompts can be customized and pre-defined for various tasks, such as automatic search and enhancements. We can give the prompts through API calls, in this way integrating the use of LLMs with our development pipeline.

Problem solving: Generative AI has been shown to be able to exhibit mathematical and algorithmic abilities, though currently still weak, because the LLMs were not trained to interpret the meaning of numeric representations and complex relationships between numbers. Programmers can describe a problem that needs to be solved and provide some guidance on how to solve it (e.g., “use dynamic programming”) and then the tool can generate code to the task. For instance, you provide a requirement like: "Make a list of major taxation laws in country X in Python language" and you will get a usable piece of code including well-named variables and documentation. For a wide variety of problems, Generative AI will spit out a usable solution, especially where commonly used algorithms are involved. Also, here we have new challenges, namely the more detailed and not trivial your request, the higher the probability that the code contains errors, which you will not find as you don’t know how to test exception cases.

Efficient development: it can be used to streamline software development processes by automating tasks such as testing, debugging, and deployment. An example could be routing of tasks such as regression test. This can help to reduce development time and costs and improve overall productivity.

Maintaining legacy. Most software that is produced today is legacy or based on legacy. A lot of trusted safety-critical software in domains such as power plants or defense is decades old. Many federal systems are based on Cobol and other antiquated languages. The challenge is increasingly to find people able to maintain such legacy. Generative AI tools in future might explain how the code works and translates it into any other language, e.g., a Python code into JavaScript. In many cases, the AI not only points out possible quality problems, but it also provides better alternatives.

Improving software quality: Generative AI can analyze large amounts of data and identify patterns that human developers might miss. An example is the selection of appropriate test cases. This can help developers to write more accurate and efficient code, and to identify and fix bugs more quickly. New challenges arise such as how to ensure that AI-based systems would not be validated by AI systems that are programmed to overlook certain defects or backdoors. Deep fake applies to software even more than only pictures.

Improving data quality: An important feature of using Generative AI for software-related tasks is the ability to fine tune an existing model on specific data. LLMs are trained on open data that is trawled from the internet, and their answers are based on what they can learn from that data. It is possible, through appropriate APIs, to give to LLMs our own data. The typical steps are to prepare our training data, then upload them to the model and let it train on that data, on top of its existing training. Then we used the fine-tuned model, which will be able to provide more relevant answers to our prompts, either typed in, or through API calls.

Achieving trust: While traditional software is based on pre-defined algorithms, current software is adaptive, self-changing and learning. Such systems do not behave according to initial specifications and might even “unlearn” what they were initially developed to do. It is not meaningful to discuss the validation of AI systems without using nearly realistic systems and contexts. Testing non-deterministic systems is difficult with deterministic tests. With traditional software testing, release criteria are based on comparing reactions to a given series of inputs with expected outputs. Simulations of cyber-physical systems suffers from the enormous state space of AI systems if not applied purposefully. For instance, autonomous vehicles need several hundred million kilometers to statistically prove that they are suitable for real traffic. Synthetic data developed by Generative AI around corner cases and critical scenarios facilitates development, testing, approval, and homologation of automatic, robotic, and autonomous systems in critical industries such as medical, aerospace, mobility and industrial production [9].

Hints for the Practitioner

While Generative AI can help companies to grow competences there are several risks that need to be considered and mitigated [7]. Technology companies and especially their venture-capitalist backers tend to just look for fast money and repeat past mistakes, namely prioritizing growth over safety. OpenAI's 2020 predecessor to ChatGPT for instance was known for "creative" outputs which were as easy to read and use as Wikipedia entries but were inhumane and racist [8]. Google explicitly announced plans to release a premature Bard, accepting the high risk it is willing to take when releasing tools based on AI technology. The lessons from social media should guide us in developing AI. What is labelled "social networks" had over the past ten years eroded true social connections and trust between people. With several hours per day on these networks mental-health and intelligence is declining at an alarming pace. Societies in many countries are deeply polarized due to fake news which is consumed without much thinking about it. Tools such as ChatGPT could further replace professional independent media and spread fake news that are neither traceable nor explainable. As developers we must get hands-on and deal with such risks.

The name of "Generative AI" cues the major pitfall. If humans rely on AI for information, it will be increasingly difficult to tell what is factual, what is an exaggerating advertisement, and what is completely made up for misinformation. The answers and solutions are generated by models based on probabilities, not necessarily found from some authoritative source. That means that they may be wrong. AI tools can *hallucinate*, responding in a wildly erroneous manner while being supremely confident that they are right. Things are improving (GPT-4 seems to be better than its predecessors), but the user should always check the answers. Relying on AI tools for tasks where you cannot determine the correct answer or how to verify it can lead to complications and pitfalls. For software development it means that human supervision and intervention is necessary, such as reviews.

With statistically driven synthesis of results, Generative AI does not have a real understanding of language. More dangerous is that it has no knowledge of the real world. The language model produces its "facts" with nice-to-read text or code. But it is the user's responsibility to verify these statements. Creativity and the capability to detect defects are the most important characteristics that distinguish you and your code from an AI. Even if it is tempting to let an algorithm do as much work for you as possible, you should always be aware that it makes mistakes, which it admittedly packages very credibly. When users started posting bug fixes generated with tools such as ChatGPT, StackOverflow banned such posts. How do they identify the fake content? The same way as a professor today must verify homework assignments, namely by means of AI. It is an arms race, and quite good tools are around to identify AI-generated documents with statistical analyses.

Practitioners should also be aware of privacy and security implications. When using a tool that analyses your code, you should be careful about what happens to that code. If your code is open source, then it probably does not matter that it may leave traces in the tool's models, or even be used as training material for the tool itself. But in proprietary code, you may not wish for your code to leave the confines of your private repositories. Different tools give different assurances for that; you should read the terms of use carefully.

It is difficult to impossible to identify what is original work and what is generated fake, such as pictures and videos used for misinformation. Demanding a source statement will not work, because those who want evil, will not follow such self-imposed rules. Watermarks on all levels or being embedded up to steganographic algorithms can easily be removed. Forensic AI researchers propose using end-to-end blockchain trust mechanisms to label what has been a proven original piece of work. Yet the challenge remains to make the initial prove on which the blockchain will be based upon.

A major risk is about cybersecurity of generated code. Generative AI tools and platform might be misused and trained to insert unwanted code fragments into any code they process. Such snippets might look innocent but could introduce backdoors, manipulate data, or feed information to external targets. Though this holds for any code reuse, cyber warfare will enter a new stage with AI-based generated code which is hard to understand and test. Verification and validation of AI will grow in relevance. Software practitioners need to enhance their competences on the right side of the "V" in order to verify accuracy of underlying AI and resulting artefacts.

Software Engineering for and with AI must start with assertions that create boundaries of what is allowed and what not. Like Isaac Asimov's robotic rules, our society and specifically IT professionals must specify upfront what is not (!) to happen. Requirements engineering must start with negative requirements, such as which misuse

cases, confuse cases and abuse cases must be avoided, and what transformations would be explicitly allows. Generative AI will in near future think and learn much more efficiently in certain areas than humans. With future AI systems rapidly improving themselves without human intervention, they could potentially wipe out humanity. A simple thought experiment would be ecology. Today many politicians put the climate change risks to extraordinary levels, forgetting about other challenges we have. A Generative AI that copies such single-minded behavior might conclude to just stop human life on earth to reduce climate change.

In a world of Generative AI and low code, it's hard to imagine a future where software engineers are as highly paid as today. Many traditional roles such as programmer will change. With the current evolution speed, we can expect that within the next three years most software companies will have an AI-augmented development and testing strategy, up from very few today. Most internet content and mobile apps will be fully or largely commanded by generative AI. Software developers will need new competencies, such as improving automatically generated software, feeding learning engines, and exploring behaviors which are not explainable. Generative AI will accelerate software development. Yet be aware of marketing hype on shortcuts to building secure, resilient software based on not deterministic technologies.

These are exciting times, not just for Software Engineers. Some even argue that we may witnessing the first sparks of Artificial General Intelligence [5] – or maybe not (yet) [6]. In any case Generative AI is here to stay, it is likely to be a game changer, and change Software Engineering as well. As with any new technology it can be used, misused, and abused. Elon Musk who is known for much, but not stopping innovations, demands oversight for artificial intelligence, having described the technology as "potentially more dangerous than nukes." As those who develop this technology, we as leading practitioners must safeguard and control AI.

Side-Bar: Generative AI Industry Case Study

At Vector we are often called to improve the quality of software systems. One typical finding is insufficient requirements and test strategy. Some software is tested several times, while some requirements and scenarios remain untested, making increasingly complex software systems impossible to trust.
While software development is simple, identifying the right requirements and test cases is a challenge for practically all companies. This is a high risk, especially when automating critical systems, such as autonomous vehicles, medical devices, and finance systems. Policy makers demand trusted AI, which demands a clear specification of intended functionality, border cases and clear demarcation lines of what must not happen. With today level of requirements engineering and test methodology we are far away from trusted AI-systems.
In such cases requirements traceability ensures that requirements are properly tracked, verified, and validated throughout the development cycle. It is perceived as highly necessary and demanded by all standards along functional safety and cybersecurity. However, in practice traceability is not maintained, and most software systems today are insufficiently tested [9]. Here are some ways in which AI can help with requirements traceability:
> Automated tagging and categorization: AI can be used to automatically tag and categorize requirements based on their type, priority, and other characteristics. This can help to ensure that requirements are properly tracked and easily searchable.
> Natural language processing: AI can be used to analyze natural language requirements and identify potential issues, such as ambiguous or conflicting requirements. It can also help to identify missing requirements or inconsistencies in the requirements documentation.
> Predictive analytics: AI can be used to analyze historical data and predict the likelihood of certain requirements being implemented successfully. This can help to identify potential risks and prioritize requirements accordingly.
> Testing automation: AI can be used to automate testing processes and ensure that each requirement is properly tested and verified. This can help to reduce the time and effort required for manual testing and improve overall testing accuracy.
The return of generative AI in connecting requirements engineering and testing comes in different currencies, namely less effort for maintaining traceability, higher product quality by test case updates even across heterogeneous tool chains, and better understanding of complex systems. The risks of generative AI remain as mentioned in the article. Be aware of your intellectual property rights and never upload software to external platforms. Do not take results of generative AI as sufficient to automate quality checks, because these tools are neither deterministic nor explainable in their chain-of-thought.

References

[1] Vaswani, Ashish, et al. "Attention Is All You Need." Advances in Neural Information Processing Systems 30 (2017): 5998-6008. https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.
[2] Devlin, Jacob, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." arXiv preprint arXiv:1810.04805 (2018). https://arxiv.org/abs/1810.04805.
[3] Radford, Alec, et al. "Improving Language Understanding by Generative Pre-Training." OpenAI Blog, June 11, 2018. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.
[4] Wolfram, Stephen. "What is ChatGPT Doing... and Why Does it Work?" (2023). https://wolfr.am/SW-ChatGPT.
[5] Bubeck, Sébastien, et al. "Sparks of Artificial General Intelligence: Early experiments with GPT-4." arXiv preprint arXiv: 2303.12712 (2023). https://arxiv.org/abs/2303.12712.
[6] Lim, Russel. "GPT-4 is Amazing but Still Struggles at High School Math Competitions." https://russelllim22.medium.com/gpt-4-is-amazing-but-still-struggles-at-high-school-math-competitions-cbc2e73738e.
[7] Ebert, C. and U.Hemel: Technology Trends: The Competence Challenge. IEEE Software, Vol. 40, No. 3, May/Jun. 2023. Digital Object Identifier 10.1109/MS.2023.3242179
[8] Chow, A.R. and B.Perrigo: "The AI Arms Race Is Changing Everything." In : Time Magazine, 17. Feb. 2023. https://time.com/6255952/ai-impact-chatgpt-microsoft-google/
[9] Ebert, C., D.Bajaj, M.Weyrich: „Testing of Software Systems". IEEE Software, Vol. 39, No. 4, pp. 8-17, Jul/Aug. 2022. https://doi.org/10.1109/MS.2022.3166755

Authors

Christof Ebert is the managing director of Vector Consulting Services, Stuttgart, 70499, Germany. He serves on the editorial board of IEEE Software and is a Senior Member of IEEE. Further information about him can be found at Linked-In: www.linkedin.com/in/christofebert . Contact him at christof.ebert@vector.com.

Panos Louridas is Associate Professor at the Department of Management Science and Technology, Athens University of Economics and Business, 10434, Athens, Greece and the Director of Research and Development at GRNET S.A., 11523, Athens Greece. Further information about him can be found at LinkedIn: https://www.linkedin.com/in/louridas/. Contact him at louridas@aueb.gr