GRAND CHALLENGES 2025

Fully Automated Software Development?

GESELLSCHAFT FÜR INFORMATIK

Developing software fully automatically — this vision no longer seems impossible with recent advances in large language models. However, its actual implementation could have consequences for society beyond those that are purely desirable. This Grand Challenge examines possible hurdles and potentials.

Software Engineering (SE) deals with the transformation of implicit and explicit knowledge about facts and requirements into intangible artifacts. These artifacts are highly formalized: in particular, programs are formal descriptions of what computers are supposed to do. Due to this high level of formalization, combined with the accessibility of artifacts to its own methods, software engineering has always been a subject of automation. A variety of tools assist with requirements management, software design, the creation of end products through complex build pipelines, and the generation and execution of tests.

Now, a new player has entered the stage of software engineering: generative AI. In particular, large language models (LLMs) contain a portion of our general knowledge and can now generate formalized outputs, such as program code, based on informally described inputs. The fact that software development is essentially a language-based transformation process makes it an ideal application for language-based generative transformers. Due to these advances, the vision of fully automatic software development no longer seems impossible.

Fully automated software engineering would have significant societal impacts: on the one hand, it could counteract the existing skills shortage and lead to higher value creation. On the other hand, it could allow anyone to generate software for their specific use case, drastically increasing digital participation. However, alongside these desirable effects, dystopian scenarios are also conceivable: automatically generated software can lead to loss of control, causing damage far beyond what we currently experience with software failures. A key challenge, therefore, is to determine and reflect on how far the automation of software development can and should go. The overarching Grand Challenge is to channel technological development in a way that ensures society only experiences the desired effects.

The Challenge

AI-based automation of software engineering (SE) is, in many ways, an ideal case for AI-driven automation overall because:

1. The field exhibits a high degree of regularity (use of standardized languages),
2. A vast, curated training base is available (software repositories, tutoring systems like StackOverflow, etc.),
3. The product is entirely intangible, meaning AI can generate and consume it without restrictions, and
4. The quality of intermediate results is objectively measurable (a program either compiles or doesn’t, tests either pass or fail), enabling automated feedback loops.

This advantageous starting position is already being exploited extensively. Current subsymbolic AI systems (such as GPT) are also being integrated with traditional systems (e.g.,

web searches). Additionally, AI systems can now be fine-tuned with non-public data or specialized for specific application domains. Considering all these developments, we can expect unprecedented advances in automated SE in the coming years.

Against this backdrop, the Grand Challenge is to channel automation in SE to achieve a socially desirable outcome. This includes efforts to reliably generate complex software with predictable quality attributes, as well as ethical reflections on what should not be automated. At a higher level, this challenge ties into the broader question of the appropriate level of collaboration between humans and AI. SE could — and probably should — take on the role of a pioneer in this regard.

Key Questions of the Grand Challenge:

1. **Ethical Reflection**: Currently, responsibility for the legal and socially acceptable behavior of software lies with humans. This is particularly relevant in unregulated contexts where ethical considerations must guide software development and deployment (see GI Ethical Guidelines) Should this responsibility shift to initiators, or must it be embedded within the automation process itself?
2. **Quality Assurance**: How can we ensure that generated products are reliable? How can existing tools, such as formal verification methods, be leveraged for AI to increase reliability and provide guarantees?
3. **Future Roles**: How will roles and responsibilities shift among stakeholders in future AI-assisted software development and operation? What are the implications for education, the economy (especially IT), society, and regulation?
4. **Managing Automation Bias**: As automatically generated results improve, people may become increasingly inclined to trust them blindly—even when they are implausible. This could become catastrophic if quality assurance is also left to automation. What absolute mechanisms of quality assurance must be implemented?
5. **Attribution and Ownership**: Compensation and liability depend on identifying who created a product. How can the origins of software products and the decisions embodied within them be securely traced? How can we ensure that human actors take responsibility for generated products?
6. **Preserving Expertise**: If AI generates software by default, how can we ensure that humans retain the expertise to perform these tasks manually? How should curricula be adapted in the future?
7. **Explainability**: In SE, stakeholders must understand that the generated system meets stated requirements, as well as additional constraints and conditions. AI explainability remains an unresolved challenge. If this issue cannot be addressed in the highly formalized domain of SE, can it be addressed at all?
8. **Usability**: Finally, the usability of AI systems in SE must be improved to achieve the intended "democratization effects."

Why We Must Address This Problem Now

The transition from an imprecise, informal, and incomplete idea to a satisfactory program is the central and challenging problem of software development. Even though statistical models such as current LLMs and the generative AI built upon them now seem very promising, it remains unclear whether and how well they can bridge this gap. Methods based on deep neural networks are black-box systems and, as of today, provide no guarantees. A

key challenge, therefore, is to integrate these remarkably powerful but potentially unreliable approaches with existing and yet-to-be-developed solutions for ensuring quality.

To fully grasp the complexity of the problem, it is also important to recognize the wide range of tasks involved in modern software engineering. These span from the initial product idea, requirement clarification, and development of domain-specific terminology to design, implementation, quality assurance, documentation, and, in some cases, system certification. Additionally, requirements will continue to evolve over time. This means that software must not only be generated at a specific point in time but must also be kept consistent alongside all associated knowledge and products despite continuous changes. The hope is that generative AI will eventually acquire or be trained with reasonable effort to develop the generalized translation and pattern recognition capabilities necessary to (fully) automate these activities. However, it remains to be analyzed whether the creative aspects of software engineering, such as innovative product development, can be meaningfully (partially) automated—and whether this is even desirable. Furthermore, how these creative ideas should be evaluated, particularly in terms of their impact on systems and society, is another open question.

Since software is continuously developed, it is rarely created from scratch but rather in the context of one or more existing systems. This means that future generative AIs must produce solutions that fit within the existing framework. A promising approach here is to make existing tools—such as those used for managing dependencies—usable for AI. However, there are currently no concrete solutions in this direction.

Another challenge is that AI must have the appropriate domain knowledge to solve a specific problem, whether for a general industry or a particular company. Today, acquiring domain expertise is already a crucial step in software development. Transferring this expertise into a highly automated process will be a challenge, especially since the knowledge base of individual companies is not included in generic LLMs. Fine-tuning and retrieval-augmented LLMs could be potential solutions. However, this again presents the challenge that fine-tuning, for instance, is not feasible for small and medium-sized enterprises due to a lack of skilled personnel, resources, and expertise. On the other hand, disclosing company knowledge into external silos is also not desirable. Avoiding a concentration of technical expertise among a few, potentially non-European, players is a broader AI challenge that extends beyond software development.

If a significantly higher level of automation than today can be achieved, software could be developed very quickly for new problems. A societal challenge will be that regulatory authorities may lack the time and expertise to establish binding rules before such systems are deployed. Experts must support legislators at the national and international levels in developing effective regulations and continuously refining them in an ongoing process.

Finally, there is the issue of acceptance: If a high degree of automation becomes possible, will it be socially acceptable for many of today's developer tasks to be automated?

Timeline for Solutions

In three to five years, we can expect highly automated generative AI solutions that support the domain-specific development of large systems. In ten years, we anticipate AI systems that take initiative, autonomously generating reliable and quality-assured software systems while being able to explain their reliability. The task of channeling this development must begin now.

These Efforts Have Already Been Invested in the Solution

Automating activities and tasks in software engineering has been a goal since the inception of the discipline. However, the gap between informally formulated or documented requirements and decisions on one side and formally described artifacts such as code on the other has long been a difficult hurdle to overcome. Proposed solutions include controlled natural language, from which further artifacts such as models can be generated, or the generation of information that assists humans in software evolution, such as traceability links (Tracelinks). Only with the recent advances in large language models has this gap begun to be bridged. So far, only individual code snippets (e.g., GitHub Copilot) or tests (e.g., CAT-LM [Rao et al., 2023]) are being generated.

Which Disciplines Must Collaborate on This Challenge

To address this grand challenge, not only must all subfields of software engineering collaborate, but also disciplines concerned with impact assessment. In particular, legal expertise is needed to address liability issues, and expertise in digital ethics and digital responsibility is required to determine the level of automation that is socially desirable [Trier et al., 2023].

Defining the Goal: Five Levels of Automation

In a Grand Challenge, it is almost by definition unclear whether it can ever be fully solved. To account for this, we define—analogous to autonomous driving—five levels of automation. The first two levels have already been partially achieved, while the fourth represents a desirable stage and thus serves as our (minimum) target scenario. It remains uncertain whether level five can be reached or if it even should be.

- Level 1: Transformation of routine tasks is automated through tools that are explicitly triggered by humans. Examples include script-driven build pipelines, requirements management (including traceability), and automatic test generation. AI systems such as recommenders are used in these processes.
- Level 2: The transformation of informal knowledge into formal artifacts is partially handled by AI, guided by human input (e.g., via prompting). Examples include ChatGPT for specification and documentation and GitHub Copilot for programming. The foundation is the general knowledge about the domain and the languages embodied in AI systems.
- Level 3: Transformation occurs based on specialized knowledge derived from an existing (partially) developed software system as context. For example, a model trained specifically on the system's codebase suggests refactorings, additions for new requirements, or bug fixes based on error reports.

- Level 4: Mixed-initiative approach: AI proactively suggests the next steps. For instance, it analyzes user behavior and app reviews to determine which functionality should be developed next.
- Level 5: Fully autonomous software development: AI makes all technical decisions and handles quality assurance independently. It interacts only with domain experts and/or end users to produce software.

The levels will not be achieved simultaneously (or possibly at all) in all areas of life. As of today, it seems more conceivable that AIs could autonomously develop apps (such as in the field of home automation) at Level 5, whereas it is less likely that safety-critical, highly interconnected systems—such as those in the energy sector—or other fundamental societal systems, like electronic court records, will be developed fully automatically in the sense described above.

Concrete target scenarios for automation and programming by end users in two exemplary areas of life are as follows. Here, the boundary between software creator and software user, as well as between classical programming and the use of intelligent systems, becomes blurred. These examples further illustrate that the interface for collaboration between humans and AI must be redesigned.

The first scenario involves programming a household robot using (spoken) everyday language. A sample dialogue might go as follows:
Human: "Robo, bring me a coffee."
Robot: "How do I do that?"
Human: "Go to the kitchen, take a cup, place it under the coffee machine spout, and press the red button. Wait until the cup is full, then bring it to me."

Another task:
Human: "Make me an omelet."
Robot: "How?"
Human: "Look up a recipe online and follow it."

Or:
Human: "Empty the dishwasher."
This presents an interesting subproblem, as the robot must know where everything belongs. Instruction for this:
Human: "Robo, open all doors and drawers in the kitchen and memorize what's inside. Later, place similar items together."
Here, the LLM's language comprehension must be mapped to the available capabilities and API of the household robot and expanded with new abilities as needed. Similar tasks could also be formulated for industrial robots, which today require extensive programming by experts.

Another scenario involves instructing autonomous vehicles. Imagine we could explain complex traffic rules—such as yielding at a bending priority road—as is done in driving school. This could be accomplished in fifteen minutes. We would no longer need 10,000 hours of example drives through such intersections until statistical learning identifies the pattern behind this traffic sign. Alternatively, we could eliminate the need for explicit

programming in these cases. Additionally, the generated program would be inspectable for verification.

Defining criteria for achieving the Grand Challenge is currently not possible, as determining what constitutes a socially desirable level of automation in software engineering remains an open discussion.

Which Social, Societal, or Economic Problems Can Be Addressed with the Grand Challenge

1. If, in the course of working on the Grand Challenge, it becomes evident that full automation cannot be achieved despite the ideal conditions mentioned above, this would have significant implications for the application of AI in other fields where conditions are less ideal. To give just one example: if the hallucinations of LLMs cannot be eliminated even under the ideal conditions found in software engineering, this would be a serious issue for other application contexts of such systems as well.
2. Accompanying automated software engineering with the questions posed by this challenge has the potential to not only theoretically but also practically eliminate the shortage of skilled professionals — especially as the demand for IT solutions increases — by clearly defining the framework for the use of highly automated processes.
3. Lowering the threshold at which IT deployment becomes economically viable leads to more inclusive access to software development (e.g., app development for NGOs, similar to RE Cares).
4. Facilitating cooperation and collaboration among various stakeholders with different backgrounds through translation of questions and answers into their respective technical languages—potentially including automated moderation (e.g., to detect and resolve contradictory requirements).
5. Raising the quality standards of "shadow IT" (e.g., Excel spreadsheets created by domain experts without IT training, which should still meet professional IT quality criteria).

The transition toward highly automated software development will occur in stages (see target scenarios). Even if not all the discussed levels are reached, progress in this direction will inevitably increase productivity in software development. Hopefully (and when properly implemented), it will also improve software quality and reliability while enabling broader participation in software development by non-experts. Advancing along these levels will thus benefit societal innovation, democratize software creation, and accelerate digital transformation.

Authors

- Prof. Dr.-Ing. Anne Koziolek
- Prof. Dr. Kurt Schneider
- Prof. Dr. Friedrich Steimann
- Prof. Dr. Walter Tichy

Supporting GI Technical Committees and Co-authors

This Grand Challenge was formulated and coordinated by members of the Technical Committee for Software Engineering. The AI, Human-Computer Interaction, Information

Systems and Computer Science and Society Technical Committees could and should continue to be involved.

Co-authors

- Prof. Dr. Gregor Engels (Universität Paderborn)
- Prof. Dr. Sabine Glesner (TU Berlin)
- Prof. Dr. Florian Matthes (TU München)
- Prof. Dr. Ralf Reussner (Karlsruher Institut für Technologie)
- Prof. Dr. Klaus Schmid (Universität Hildesheim)

Supporters

- Prof. Dr. Colin Atkinson (Universität Mannheim)
- Prof. Dr. Steffen Becker (Universität Stuttgart)
- Prof. Dr. Thorsten Berger (Ruhr-Universität Bochum)
- Prof. Dr. Dirk Beyer (Ludwig-Maximilians-Universität München)
- Prof. Dr. Eric Bodden (Universität Paderborn)
- Prof. Dr. Bernd Brügge (TU München)
- Prof. Dr. sc. Maria Christakis (TU Wien)
- Prof. Dr. Michael Felderer (Deutsches Zentrum für Luft- und Raumfahrt (DLR) und Universität zu Köln)
- Prof. Dr. Martin Glinz (Universität Zürich)
- Prof. Dr. Falk Howar (TU Dortmund)
- Prof. Dr. Jan Jürjens (Universität Koblenz)
- Jun.-Prof. Dr. Verena Klös (TU Dresden)
- Prof. Dr. Samuel Kounev (Universität Würzburg)
- Dr. Heiko Koziolek (ABB Corporate Research)
- Prof. Dr. Leen Lambers (BTU Cottbus-Senftenberg)
- Prof. Dr. Anna-Lena Lamprecht (Universität Potsdam)
- Prof. Dr. Stefan Leue (Universität Konstanz)
- Prof. Dr. Daniel Mendez (fortiss GmbH)
- Prof. Dr. Mira Mezini (TU Darmstadt)
- Prof. Dr. Barbara Paech (Universität Heidelberg)
- Prof. Dr. sc. Michael Pradel (Universität Stuttgart)
- Prof. Dr. Rick Rabiser (Johannes Kepler Universität Linz)
- Prof. Dr. Albrecht Schmidt (Ludwig-Maximilians-Universität München)
- Jun.-Prof. Maike Schwammberger (Karlsruher Institut für Technologie)
- Prof. Dr. Janet Siegmund (TU Chemnitz)
- Prof. Dr. Norbert Siegmund (Universität Leipzig)
- Prof. Dr. Wolfgang Reif (Universität Augsburg)
- Prof. Dr. Gabriele Taentzer (Universität Magdeburg)
- Prof. Dr. Matthias Tichy (Universität Ulm)
- Prof. Dr. Andreas Vogelsang (Universität Köln)
- Prof. Dr. Stefan Wagner (Universität Stuttgart)
- Prof. Dr. Manuel Wimmer (Johannes Kepler Universität Linz)

Further reading

Rao, N., Jain, K., Alon, U., Le Goues, C., Hellendoorn, V.J. (2023, October). CAT-LM: Training Language Models on Aligned Code And Tests. In Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (to appear, preprint via https://conf.researchr.org/home/ase-2023)

Trier, M., Kundisch, D., Beverungen, D., Müller, O., Schryen, G., Mirbabaie, M., & Trang, S. (2023). Digital Responsibility: A Multilevel Framework for Responsible Digitalization. Business & Information Systems Engineering, 65(4), 463-474.