JoinedWorkz

Use AI broadly in software development while reliably keeping your technical standards.

JoinedWorkz combines AI-assisted development with a modeling language shaped by customer projects and with versioned code generators. Developers and AI describe the application in a compact model; generators produce the recurring part of backend and frontend from it.

AI-assisted development with models and generators

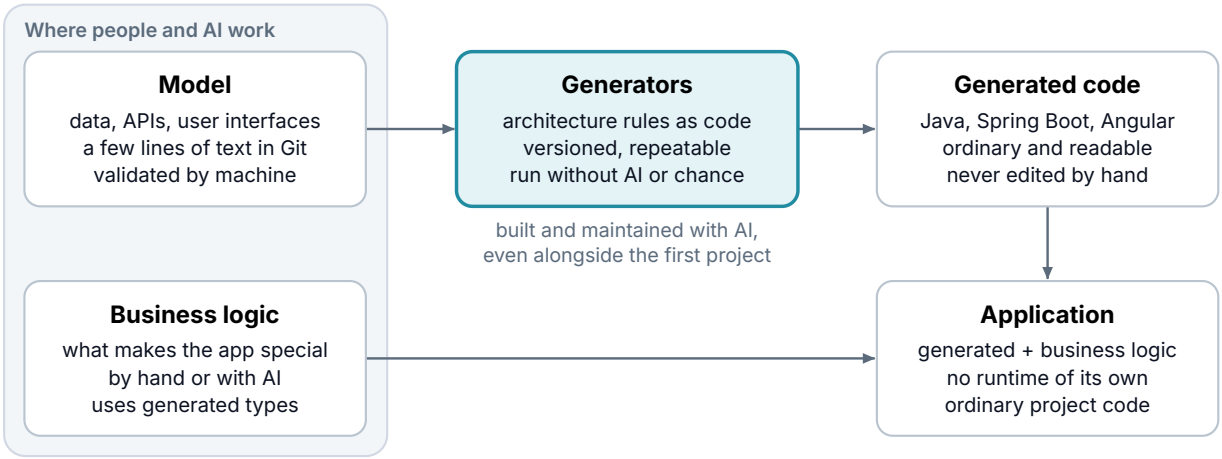
THE PROBLEM

AI writes code in minutes. In long-lived enterprise applications, the bottleneck therefore shifts to reviewing and maintaining it:

- **More speed, less stability**
  As AI adoption grows, delivery throughput rises, and so does delivery instability. Time saved writing code is often spent verifying it (DORA).
- **More copies, less upkeep**
  An analysis of 623 million code changes shows 81% more duplicated code than before AI tools; the metric for refactoring fell by 70% (GitClear, GitKraken).
- **Standards by probability**
  An AI agent follows technical rules if it knows them and keeps them in context. That is not guaranteed, least of all across many changes and teams.
- **Standard code is written again and again**
  Even with AI, recurring code is rewritten, reviewed and paid for again with every feature.

THE APPROACH

JoinedWorkz separates what repeats from what makes an application special. The model describes data, APIs, components and user interfaces as text. Versioned generators turn it into ordinary, readable code. Business logic is still written by hand or with AI and is kept separate from the generated code.



The model remains the authoritative source for the parts generated from it: changes are made in the model or in the generator. The same model and the same generator version produce the same code.

HOW AI AND JOINEDWORKZ WORK TOGETHER

AI contributes in three places – and deliberately not in a fourth:

1. **AI writes the model**
   Agents write and change the model: a few lines in a language with fixed terms. Types, references and rules are validated by machine before any code is generated.
2. **AI develops the business logic**
   Custom calculations, special user interfaces and integrations live in separately maintained code. Generated types and APIs set the frame.
3. **AI helps build the generators**
   Architecture knowledge goes into reusable generators. AI helps to implement them, test them against examples and evolve them.
4. **AI does not write the standard part**
   It is generated without a language-model call, following versioned rules. Patterns developed and verified once apply to every new feature.

AN EXAMPLE IN NUMBERS

22 → 532
model lines for one edit page produce 532 lines of code in 5 files.

3,400 → 55,000
model lines of a sample application with 84 screens produce about 55,000 lines of code. For an agent: roughly 30,000 instead of 450,000 tokens.

69 → 1
deprecated framework calls in generated code, replaced by one fix in the generator.

Own measurements on the sample project of the Angular generator, November 2026; token figures estimated. Lines are not a measure of value; the numbers show the order of magnitude. A publicly reproducible backend example is in the documentation.

STILL VALUABLE AS AI IMPROVES

- **Rules apply, not just most of the time**
  Rules an agent receives as text are followed most of the time. Rules in the generator always apply to the generated part.
- **Decisions are preserved**
  Architecture decisions are not made anew in every AI session; they are applied again with every generation run.
- **Knowledge stays when AI models change**
  Experience is captured in the modeling language, the implementation in the generators. Both are versioned and independent of the AI model.
- **Better AI makes better use of the foundation**
  It models more reliably and evolves generators faster. The value of reuse remains.

BENEFITS OVER THE LIFECYCLE

- **Evolve consistently**
  Data structures, APIs and user interfaces refer to shared definitions. Backend and frontend stay in sync.
- **Improve centrally**
  A better pattern or a modernized framework call is changed once in the generator and adopted in a controlled way.
- **Focus the review effort**
  Reviews concentrate on model, generators and business logic. The finished application is still tested as a whole.
- **Faster onboarding**
  New team members and AI agents find the same patterns in every application.
- **Use AI where it counts**
  For modeled changes an agent needs less context. Generation itself runs without a language model.
- **Stay in your own technology stack**
  The result is ordinary code with no JoinedWorkz runtime. If you stop generating, you carry on with it as normal code, with or without AI.

EXTENDING GENERATORS WITH AI

Companies can capture their own architecture knowledge in reusable generators:

No finished generator required

- You start where suitable building blocks exist. Missing capabilities and company-specific rules are added step by step.
- For a new technology stack, the generator can be built with AI support alongside the first project – ideal when several applications share the same architecture.
- The first application carries part of the setup; each further one benefits more. Scope, responsibility and maintenance are agreed deliberately.

Even where AI use in the project is restricted

- If a customer does not allow powerful public AI models for project content, the service provider can still develop and test the generators with AI: on neutral sample models.
- Project models and the code generated from them are processed inside the customer environment. The customer's rules remain decisive.

PREREQUISITES AND LIMITS

JoinedWorkz is a method with tools, not a silver bullet:

- **The project has to back the approach**
  Client, architects and team choose it deliberately; build and ways of working are adapted to it. The method only works if the project carries it.
- **Repetition pays off**
  It suits long-lived business applications and several applications with the same architecture. For small one-off projects, implementing directly with AI is often simpler.
- **Only the standard part is generated**
  Business logic and highly individual user interfaces remain development work. Generator quality is decisive: errors and fixes alike take effect everywhere.
- **The effort belongs in the calculation**
  Modeling and generator maintenance take time; the savings depend on the project. Agents need guidance through documentation and AI context.

CURRENT STATUS

JoinedWorkz is used in customer projects; references cannot be named publicly yet. Stable generators are available for Java and Spring Boot. The Angular generator is in use in one customer project and about to be published. The approach includes the modeling language CMN, the editor JoinedWorkz Studio with model validation, integration of generation into the Maven build, and documentation and AI context for agents. Only the editor requires a license.

CONTACT

JoinedSystems develops JoinedWorkz and supports its introduction, the adaptation of generators, and the work of teams and AI agents with the models. Contact: Karl Hönninger.

https://joinedsystems.eu info@joinedsystems.eu https://www.joinedworkz.org

As of November 2026 · Sources: DORA, "Balancing AI tensions" · LeadDev, "Code maintainability plummets in the AI coding era"