Future of Software Engineering

Towards Specification Engineering

Jürgen Döllner

Hasso Plattner Institute
University of Potsdam

March 5, 2026

Software Engineering Since the 1950s

Human-centered development processes

- Programming models and languages
- System abstractions and architecture
- Development methodologies (Waterfall, Agile, DevOps)
- Specialized roles (architects, developers, testers, operations)
- Quality assurance (testing, reviews, verification)
- Tooling and infrastructure (IDEs, CI/CD, version control)

A Structural Shift

Automation in Software Engineering is not new

- Compilers, build systems, Continuous Integration / Delivery
- Model-driven engineering, generators, domain-specific languages
- Static analysis, automated testing and verification

What is new

- Machines can interpret informal and semi-formal specifications
- Machines can synthesize system structure, architecture, and code
- Machines can iteratively modify systems based on feedback and constraints

Implication

Software engineering undergoes a shift from *human-centered processes* to *hybrid human–machine processes*.

From Coding to Specification Engineering

Traditional Software Engineering	Emerging Software Engineering
implementation-centric	specification-centric
humans implement systems	machines generate system implementations
testing validates code	verification governs generation

Hypothesis

Software engineering transforms into *specification engineering*.

Case Study: A Domain-Specific Stress Test

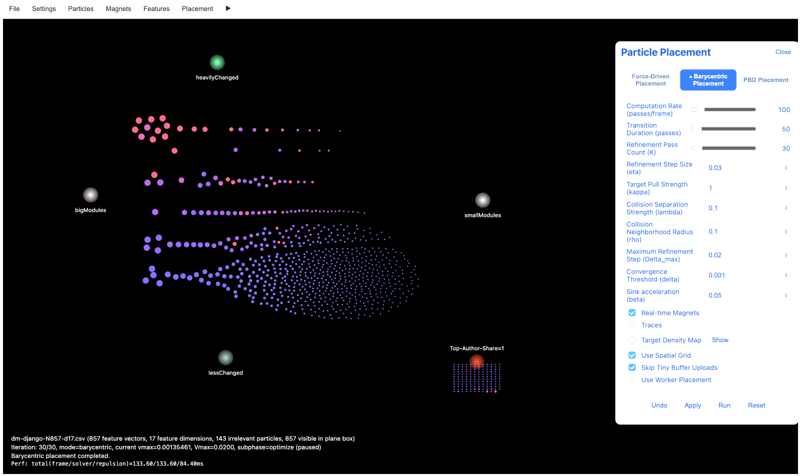
Personal experiment (7 days, ∼75% effort)

- Target: an app for visual exploration of high-dimensional feature spaces
- Platform: HTML / JavaScript with WebGL rendering
- Algorithmic depth: computational geometry, physics-based modeling, interactive rendering
- Domain-specific problem (not a standard CRUD/business application)

Outcome

- AI-assisted generation produced ∼50 000 lines of code
- Complete interactive system: UI, optimized rendering, serialization, synthetic test data
- Integrated and extensive automated test suite

Case Study: A Domain-Specific Stress Test



Particle Placement

Computation Rate (passes/frame)	100
Transition Duration (passes)	50
Refinement Pass Count (K)	30
Refinement Step Size (eta)	0.03
Target Pull Strength (kappa)	1
Collision Separation Strength (lambda)	0.1
Collision Neighborhood Radius (rho)	0.1
Maximum Refinement Step (Delta_max)	0.02
Convergence Threshold (delta)	0.001
Sink acceleration (beta)	0.05

- ✓ Real-time Magnets
- ☐ Traces
- ☐ Target Density Map Show
- ✓ Use Spatial Grid
- ✓ Skip Tiny Buffer Uploads
- ☐ Use Worker Placement

```
dm-django-N857-d17.csv (857 feature vectors, 17 feature dimensions, 143 irrelevant particles, 857 visible in plane box)
Iteration: 30/30, mode=barycentric, current vmax=0.00135461, Vmax=0.0200, subphase=optimize (paused)
Barycentric placement completed.
Perf: total(frame/solver/repulsion)=133.60/133.60/84.40ms
```

Figure: Example: feature space visualization of the Django framework. Each module is represented as a feature vector derived from repository metadata (e.g., lines-of-code, churn, commits).

Case Study: A Domain-Specific Stress Test

Specification-driven development

develop theoretical specification (e.g., Overleaf)

↓

map specification to implementation and modify the system (Codex)

↓

testing and evaluation

↓

next iteration

Key observation

Very short cycles between theoretical advances and implementation provide immediate feedback and enable rapid revision of conceptual ideas.

Emerging Core Activities

Future software engineering focuses on three activities

- **Specification engineering**
  describing systems, behaviors, architectures, and constraints
- **Generation control**
  guiding and calibrating machine-based development processes
- **Verification and testing**
  ensuring correctness of generated systems at micro-, meso-, and macro levels

Testing Becomes Central

Testing and verification become core disciplines in generative software development

- machines rapidly modify, extend, or refactor existing source code
- changes may affect large parts of a system simultaneously
- verification becomes the bottleneck

Ensuring correctness of generated systems at micro-, meso-, and macro levels.

Implications for Organizations

Preparing for hybrid human–machine software engineering

- **Strengthen specification practices**
  precise system descriptions and systematic specification methodologies
  specifications and their evolution become key assets of software systems
- **Establish generation workflows**
  integrate machine-based code generation into development processes
- **Strengthen verification and human-in-the-loop control**
  automated testing across micro-, meso-, and macro levels
  continuous observation and evaluation of generated systems

Monitoring System Evolution

Keeping humans in control of generative software development

- **Transparent system evolution**
  continuous monitoring of system modifications and generated artifacts
  comparison of alternative generation trajectories
  controlling proliferation of system components and generated code
- **Explainable generation processes**
  traceability of how specifications lead to architectural and code changes
  explainability of system state, quality, correctness, and risks
- **Detection of deviations from expected behavior**
  identifying unexpected, unsafe, or unintended system behavior
  forecasting when human intervention becomes necessary

Human control requires transparency and explainability.

Conclusion: A Change of Mindset

Software engineering is entering a new phase

humans as programmers

$\downarrow$

humans as system specifiers and supervisors

machines implement and evolve software systems

Software engineering becomes specification engineering