No End of Programming

Jürgen Döllner

Hasso-Plattner-Institute

June 4, 2025

Introduction

- Generative AI is fundamentally changing the way software has been developed since the 1950s.
- This change is so significant that some are even suggesting that the *current model of software development* is coming to an end.
- What does it mean in the long term when machines program machines?

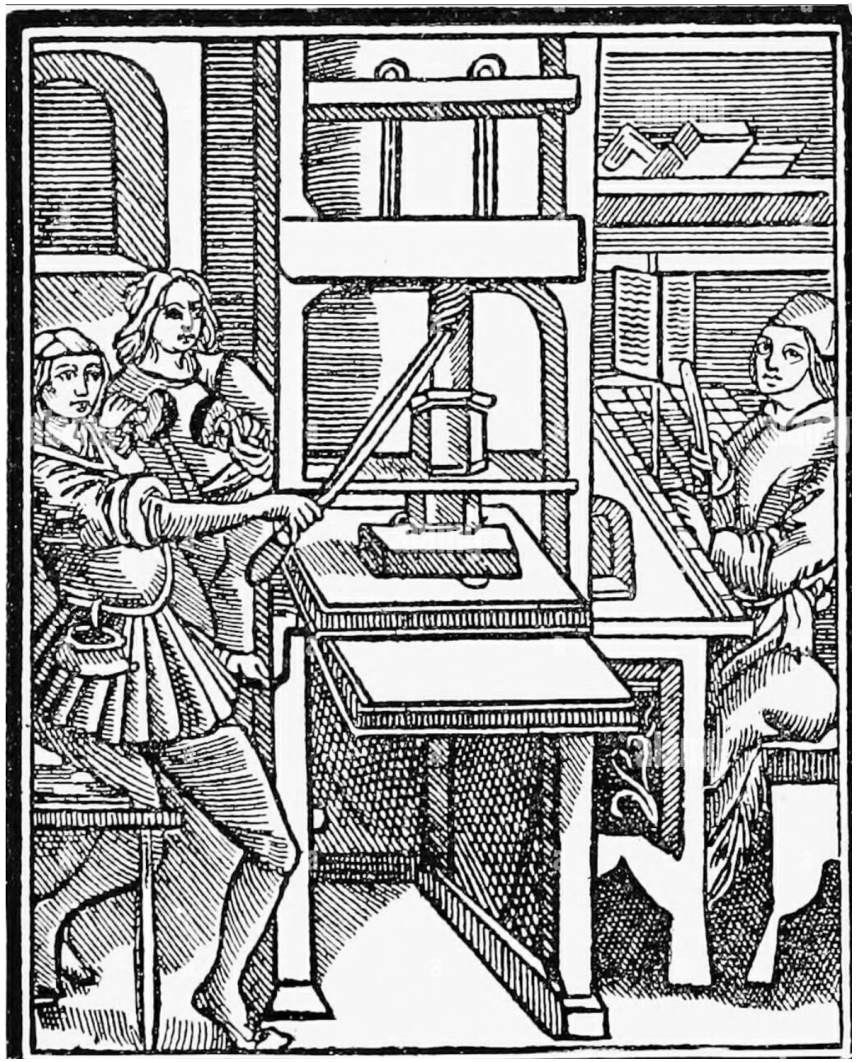
Introduction

- "Programming will be obsolete. I believe the conventional idea of 'writing a program' is headed for extinction, and indeed, for all but very specialized applications, most software, as we know it, will be replaced by AI systems that are *trained* rather than *programmed*.

Disruptive Innovations

Scriptorium Era: The illustration shows work in a scriptorium using the example of the 'Miracles de Notre Dame' (1462) for manual book production.

Disruptive Innovation



Gutenberg Bookpress: The illustration shows Gutenberg's first book press, which made mass production of printed information possible for the first time. This triggered an unimaginable boom in business and science.

Disruptive Innovations

```
%!PS-Adobe-3.0
%%Creator: Wolfram Mathematica 11.0.1.0 for Microsoft Windows (32-bit)
%%CreationDate: Fri Mar 30 14:17:02 2018
%%Pages: 7
%%DocumentData: Clean7Bit
%%LanguageLevel: 3
%%DocumentMedia: A4 595 842 0 () ()
%%BoundingBox: 54 33 539 812
%%EndComments
%%BeginProlog
/languagelevel where
{ pop languagelevel } { 1 } ifelse
3 lt { /Helvetica findfont 12 scalefont setfont 50 500 moveto
  (This print job requires a PostScript Language Level 3 printer.) show
  showpage quit } if
/q { gsave } bind def
/Q { grestore } bind def
/cm { 6 array astore concat } bind def
/w { setlinewidth } bind def
/J { setlinecap } bind def
/j { setlinejoin } bind def
/M { setmiterlimit } bind def
/d { setdash } bind def
/m { moveto } bind def
/l { lineto } bind def
/c { curveto } bind def
/h { closepath } bind def
/re { exch dup neg 3 1 roll 5 3 roll moveto 0 rlineto
      0 exch rlineto 0 rlineto closepath } bind def
/S { stroke } bind def
```

Postscript and Digital Printing: PostScript as a universal description language for page-structured 2D information (John Warnock, Adobe, 1982) – silent omnipresence. PostScript and PDF democratized access to information, removed technical barriers, and ensured that documents could be created, shared, and printed anywhere in the world with absolute fidelity.

Disruptive Innovations

- Initially limited performance compared to established solutions, but foreseeably cheaper or simpler
- Scaling and mass adoption as the technology matures

- Radical change to existing processes, products or technologies
- Development of new target groups and markets that were not previously served

- Long-term, far-reaching displacement of existing markets and society (e.g., democratization of knowledge)

Naturalness of Software

On the Naturalness of Software

Abram Hindle, Earl Barr, Mark Gabel, Zhendong Su, Prem Devanbu
devanbu@cs.ucdavis.edu

Unpublished version of ICSE 2012 paper, with expanded future work section
Enjoy! Comments Welcome.

***Abstract*—Natural languages like English are rich, complex, and powerful. The highly creative and graceful use of languages like English and Tamil, by masters like Shakespeare and Avvaiyar, can certainly delight and inspire. But in practice, given cognitive constraints and the exigencies of daily life, most human utterances are far simpler and much more repetitive and predictable. In fact, these utterances can be very usefully modeled using modern statistical methods. This fact has led to the phenomenal success of statistical approaches to speech recognition, natural language translation, question-answering, and text mining and comprehension.**

We begin with the conjecture that *most software is also natural*, in the sense that it is created by humans at work, with all the attendant constraints and limitations—and thus, like natural language, it is also likely to be repetitive and predictable. We then proceed to ask whether a) code can be usefully modeled by statistical language models and b) such models can be leveraged to support software engineers. Using the widely adopted n-gram model, we provide empirical evidence supportive of a positive answer to both these questions. We show that code is also very repetitive, and in fact even more so than natural languages. As an example use of the model, we have developed a simple code completion engine for Java that, despite its simplicity, already improves Eclipse's completion capability. We conclude the paper by laying out a vision for future research in this area.

what people actually write or say. In the 1980's, a fundamental shift to *corpus-based, statistically rigorous* methods occurred. The availability of large, on-line corpora of natural language text, including "aligned" text with translations in multiple languages[1], along with the computational muscle (CPU speed, primary and secondary storage) to estimate robust statistical models over very large data sets has led to stunning progress and widely-available practical applications, such as statistical translation used by `translate.google.com`.[2] We argue that an essential fact underlying this modern, exciting phase of NLP is this: *natural language may be complex and admit a great wealth of expression, but what people write and say is largely regular and predictable*.

Our central hypothesis is that the same argument applies to software:

> *Programming languages, in theory, are complex, flexible and powerful, but the programs that real people actually write are mostly simple and rather repetitive, and thus they have usefully predictable statistical properties that can be captured in statistical language models*

Naturalness of Software

> *Programming languages, in theory, are complex, flexible and powerful, but the programs that real people actually write are mostly simple and rather repetitive, and thus they have usefully predictable statistical properties that can be captured in statistical language models and leveraged for software engineering tasks.*

- Statistical regularities $\Rightarrow$ predictability of code

Generative AI

Programming Languages are Languages

- Well-defined syntax and semantics
- Known, common programming patterns
- Limited vocabularies
- Enhanced learning due to coding standards and conventions

LLMs Trained on Code

- Exposure to vast codebases allows LLMs to learn and understand source codes, APIs, frameworks, etc.
- Complementary information steams from topics over programming (e.g., stackoverflow, ticket systems)
- Structured code aligns with LLM sequence modeling
- AI models learn from both code and natural language, enhancing versatility

Generative AI: Impact on Software Development

Shift from Coding to Solution Design

- Generative AI shifts focus on specifying *what* a solution should do, not *how* to implement it.

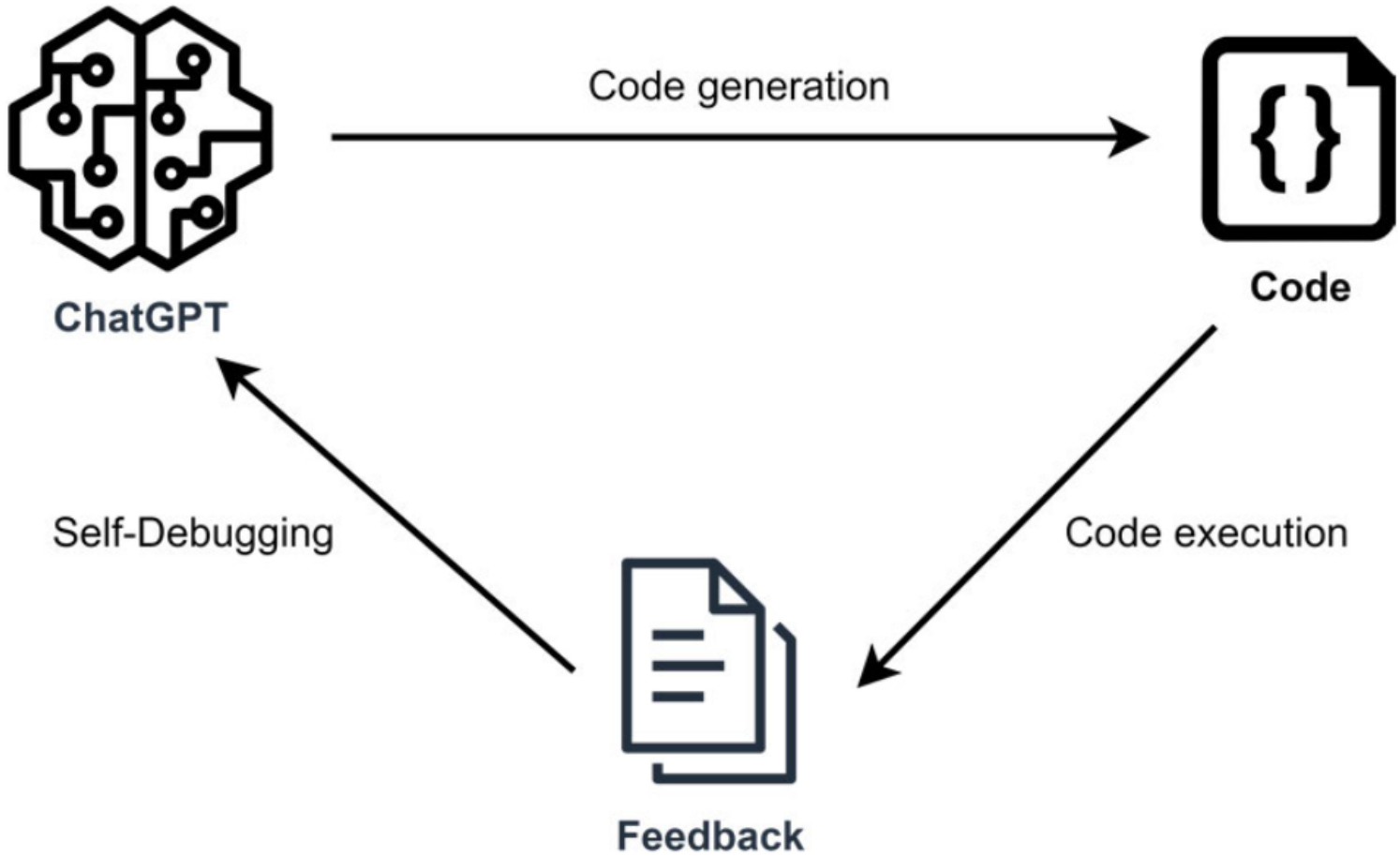
- Software developers spend a large part of time on design, coding, debugging, redesign, and testing code as well as on maintenance, refactoring, etc.
  ⇒ Generative AI automates large parts of these activities.
  ⇒ Time required for coding will ultimately become insignificant.

Generative AI: Impact on Software Development

Vision: Enabling Real-Time Software Development On-Demand

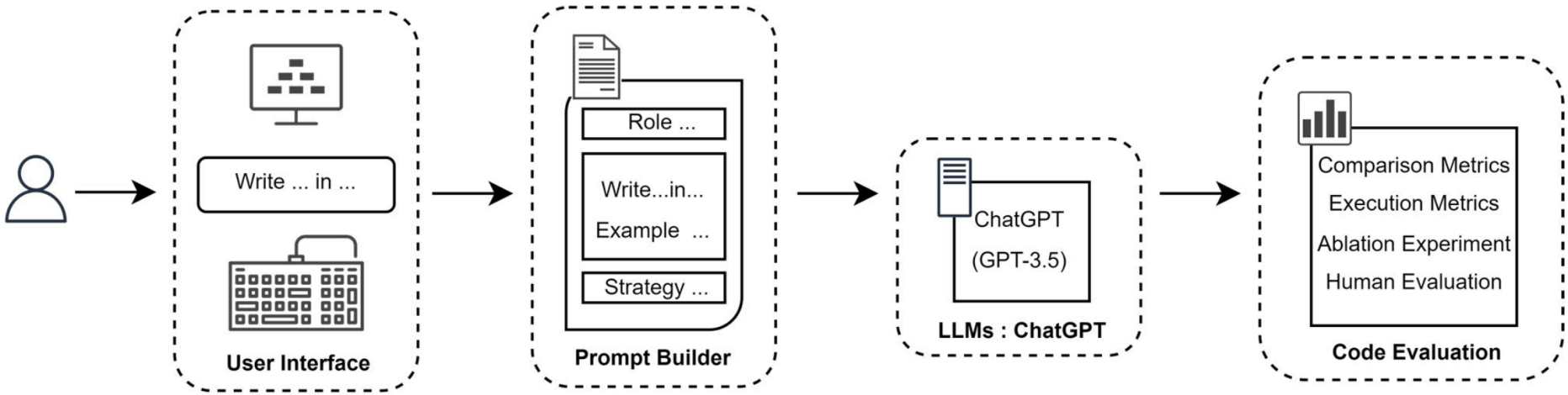
- If software can be automatically, precisely generated within a controlled environment and a given context, it can be generated on demand, i.e., just-in-time.
- For example, coding, testing, deployment and execution of software would only be performed once, exactly when a service receives a specific request.
- It will be no longer necessary to develop a general, all-encompassing software system in advance when a specific software system could be derived at any time.
- Example: Exploratory data analysis by ChatGPT.

Excursion: Self-Debugging



Y. Li, J. Shi, Z. Zhang: An Approach for Rapid Source Code Development Based on ChatGPT and Prompt Engineering, IEEE Access, 2024

Excursion: Prompt Engineering



Y. Li, J. Shi, Z. Zhang: An Approach for Rapid Source Code Development Based on ChatGPT and Prompt Engineering, IEEE Access, 2024

Excursion: Prompt Engineering

Generative AI: No Crystal Ball

- Unspecified aspects become entropy, the solver must *guess* or *default*.
- **Under-Specification**
  - Ambiguous goals, missing constraints, hidden assumptions
  - Combinatorial explosion of candidate programs; unpredictable behavior
- **Over-Specification**
  - Premature design decisions, irrelevant or excessive constraints
  - Narrows to one brittle solution; blocks optimization or innovation
- Both extremes inflate risks and costs; the goal is a *just-enough* specification that leads to a stable, correct implementation.

No End of Programming

Selected Research Topics

- Prompt Management and prompt engineering
- Agent-based software development mechanics
- Ensuring deterministic code generation
- Role of high-level system architecture specification
- Library design and implementation
- API design towards LLMs
- ...

Contact Information:

- Prof. Dr. rer. nat. habil. Jürgen Döllner
  Hasso-Plattner-Institute for Digial Engineering
  University of Potsdam, Germany
- Email: doellner@hpi.de
- Website: `https://www.hpi.de/doellner`