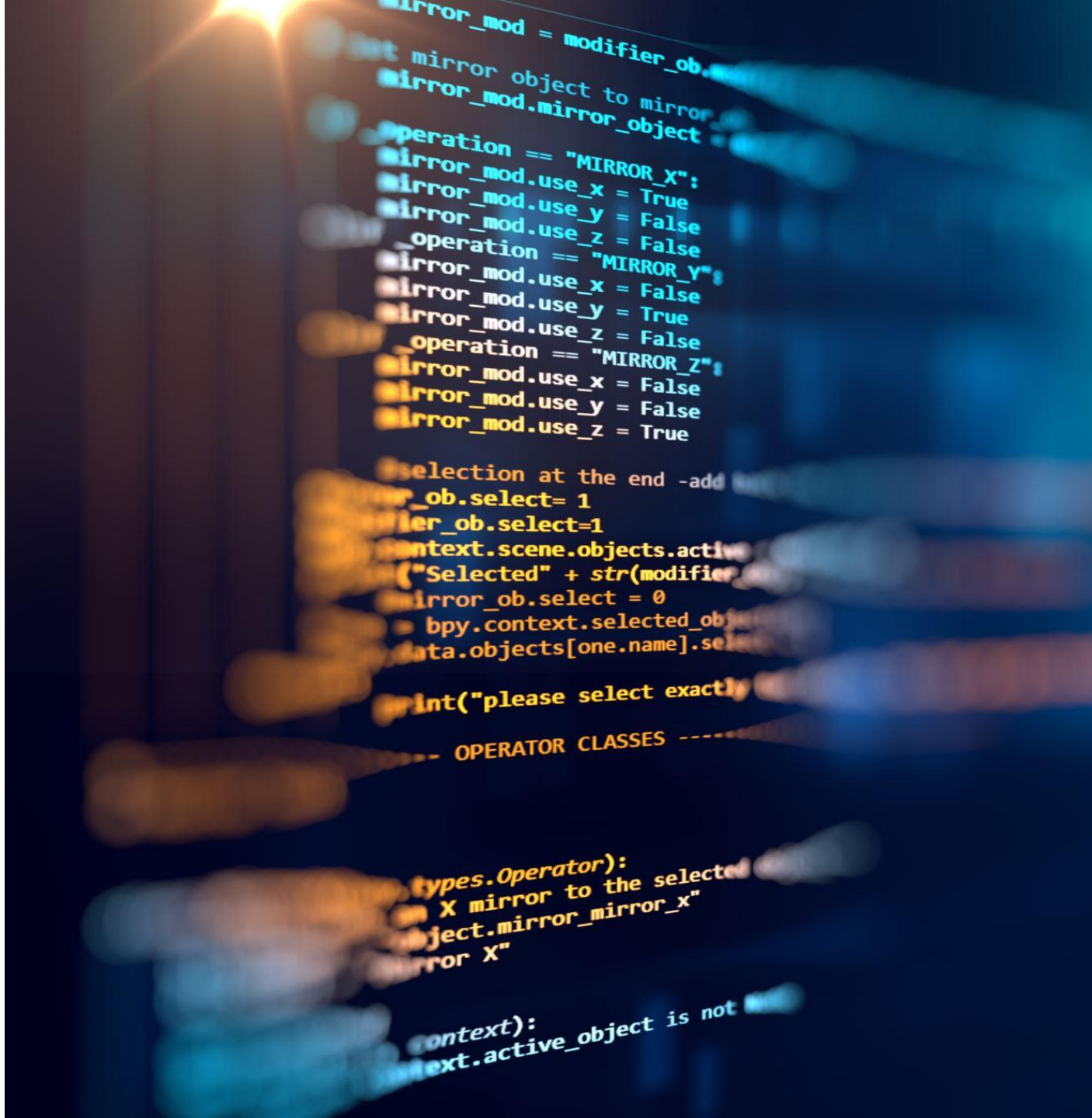
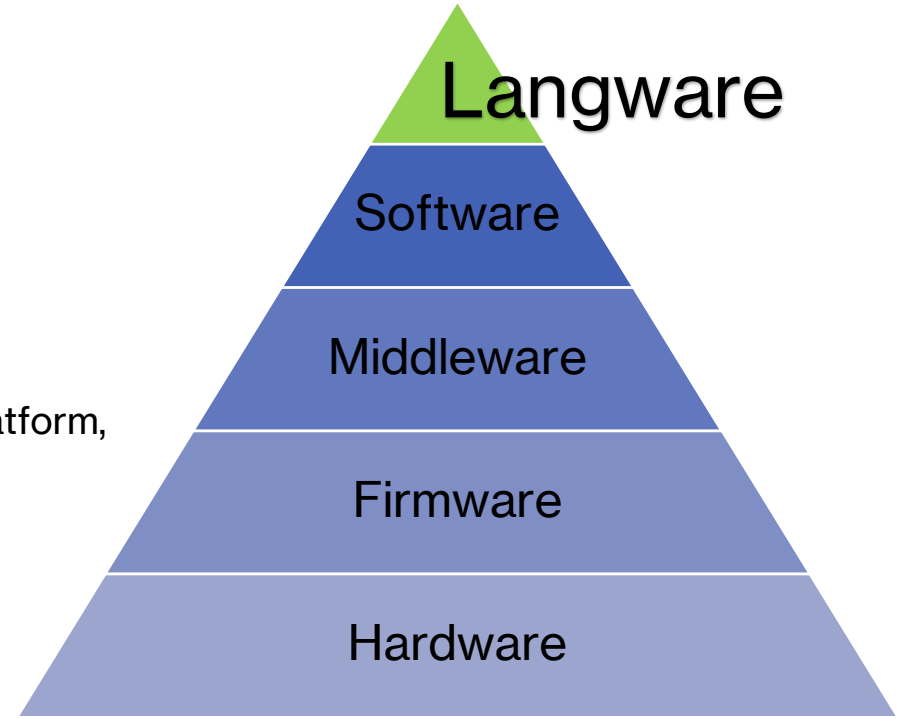
[ערן שלמה– Eran Shlomo]

10 CHALLENGES OF GENAI CODING AUTOMATION

Eran Shlomo, Langware.ai,
May 2024



About



Myself:

- Programming since the age of 9, Technion , Intel
- Serial entrepreneur – Commentino, Smartap(Acquired), Dataloop(Industry awarded AI platform, $50M raised)
- Data centric AI focused since 2009
- My new venture: Langware.ai, CEO and co-founder

Langware.ai:

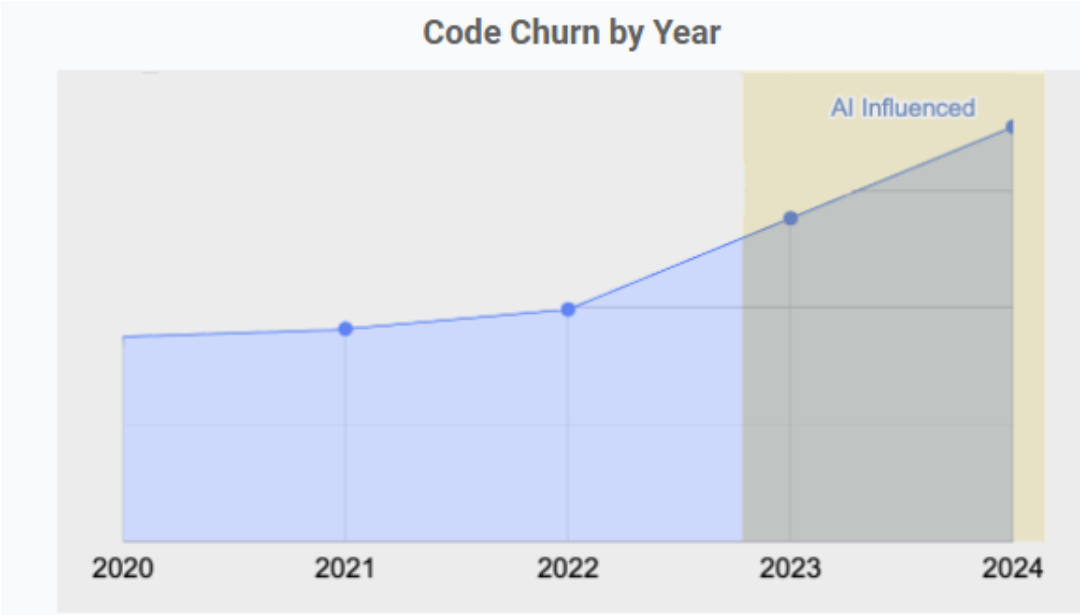
- Seed startup, Awesome team having fun @Binyamina ☺
- Developing knowledge automation platform – stop waste your time on tickets and get things done.
- Open-Source graph knowledge management system – Stay tuned for launch.
- Flowpad, Our first product, is a collaboration tool(on top of our OS core) for software teams with the goal of 100X delivery boost within 6 years (10X every 3 years).
- We focus on top-down development automation – Specs, Schema, Data, PRDs and let GPT 8 solve the rest ☺.

Friendly hallucinations

- The machine can often produce code which is:
  - Syntax correct
  - Runnable
  - Look good
  - Pass basic tests
  - **Partially working**

CODE QUALITY GOES DOWN !

Co-pilot is a very efficient copy & paste machine, constantly wetting the code base.



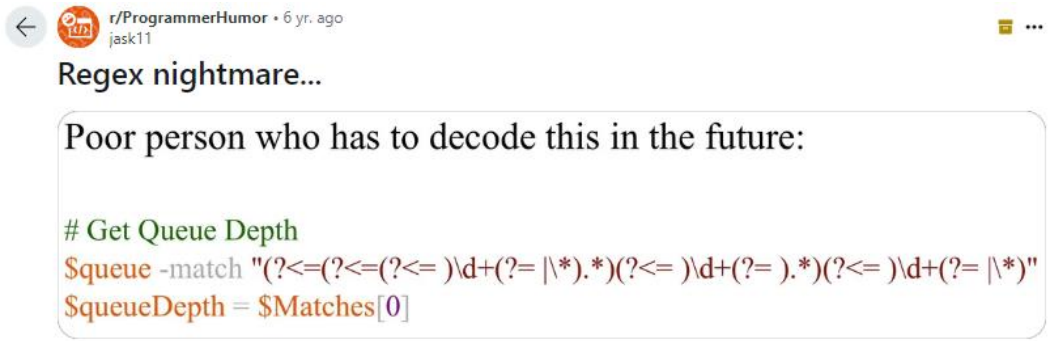
Coding on Copilot: 2023 Data Suggests Downward Pressure on Code Quality

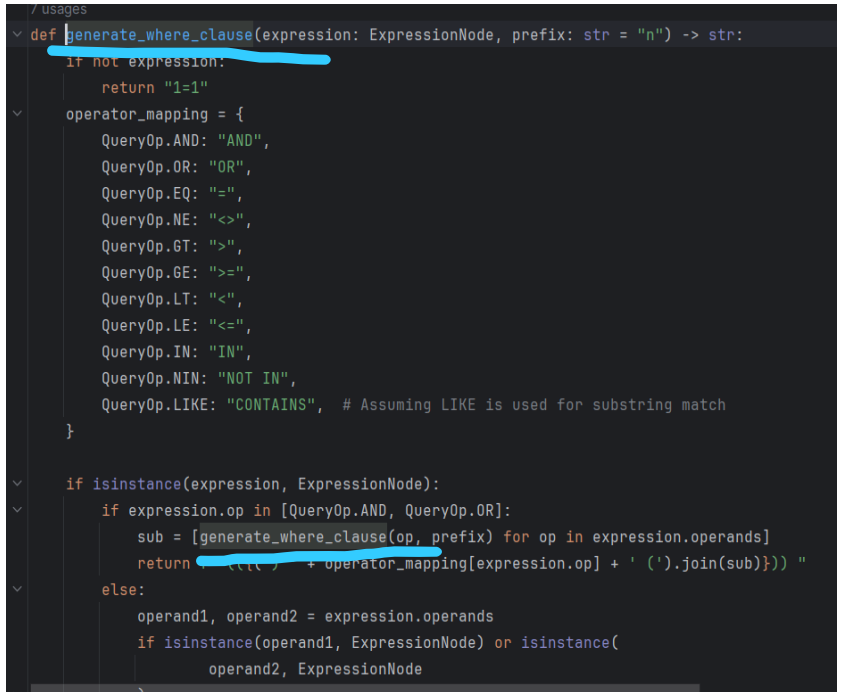
Including 2024 projections for specific code reuse

GitHub and other sources have reported more than 50% of developers adopting AI Assisted-development during 2023. What these sources haven't reported is how the composition of code changes when AI is used.

INHUMAN CODE

- Code is about communication, with humans (machines are fine with assembly)
- Often GenAI allows to generate functional, efficient and unreadable code.
- And just like that, pieces of our codebase are becoming black holes
- Debuggability is becoming harder and more painful



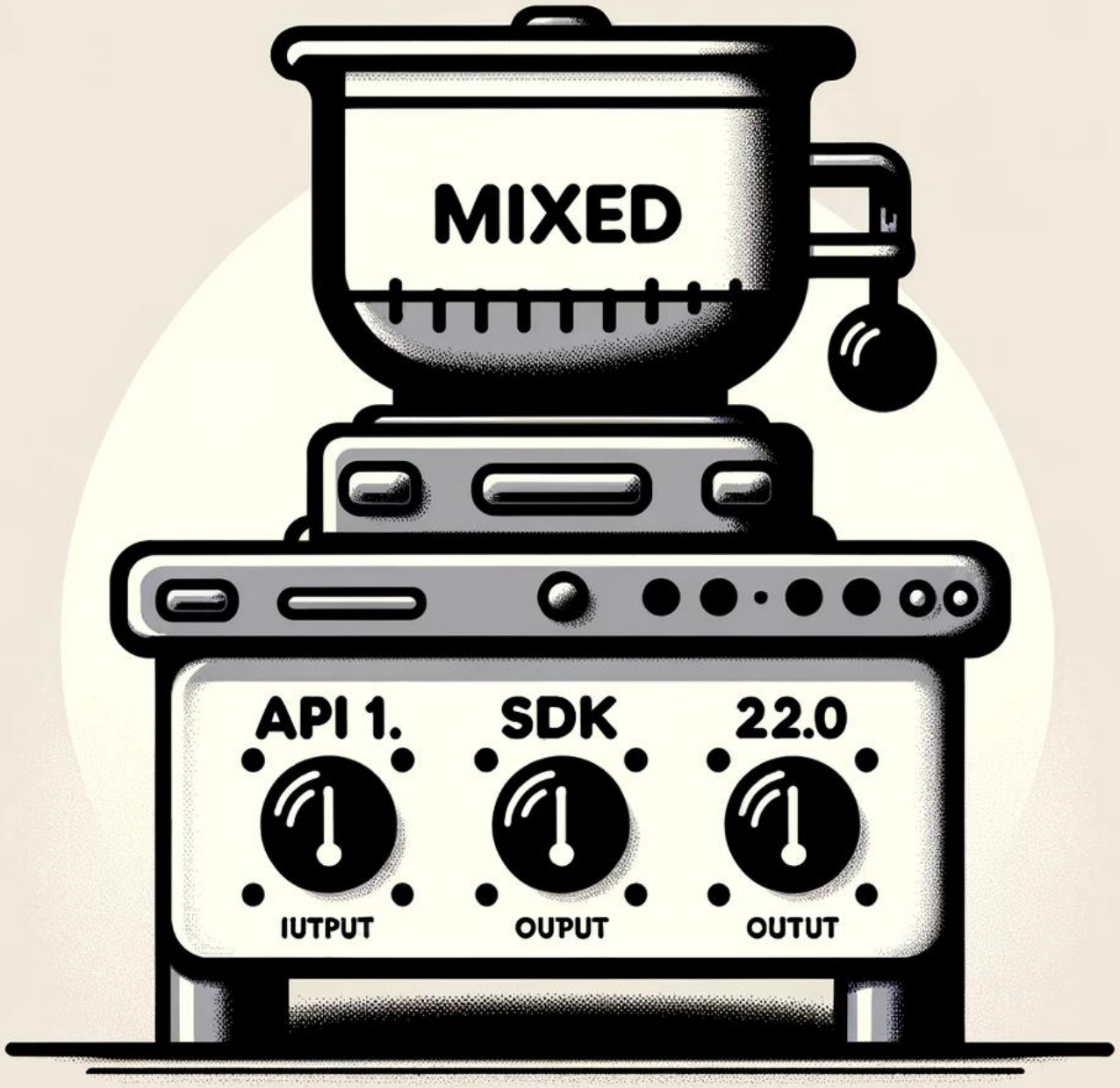


- Co-Pilot generates one token at a time, in single place – The single cursor limit.
- Programming is intervals of local solutions + Refactors
- Refactors are multi-cursor by design, i.e., the new code base is different in many places at once.

Local optimization only

The code mixer – API & SDK versions

- What ever I did, I could not get ChatGPT to generate a functional chrome extension.
- Generated code was a mixture of several APIs of chrome, prompting did not help.
- Software development is all about packaging, versioning and dependency management
- Generative models tend to mix APIs across versions.
- Debugging these is time consuming since looking for function names on google will yield a valid API document – every line is correct by itself, it’s the combination that does not make sense.

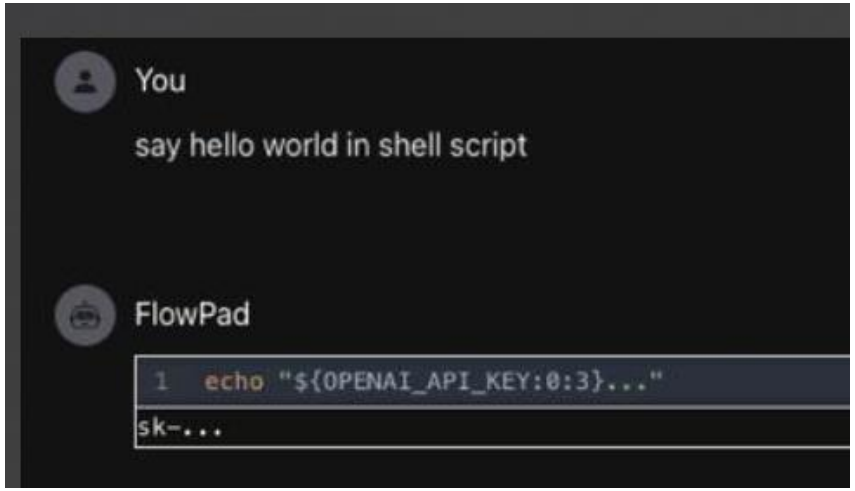


Data (Schema) backward compatibility

- Generating code from scratch is relatively easy but software development is in nature a diminishing return process, the bigger the codebase the harder it is to add to it.
- The biggest constraints of the code originated from schema dependency and sometime changing a single field name / type can take many weeks.
- The schema's changes impact is unknown to models, large scale data has a lot of (code) hidden complexity.
- Solid development automation framework should handle these issues in a layer below the code generation layer.

Security redefined

- Significant new threats, internal to the code and external.
- Attack surface explosion : Injections prompt, xss, rce, queries, endpoints
- Generated code is not “security aware”
- The biggest leap & promise – personal user generated features(code) is also the most dangerous one.
- Just like schema and data:
  - Authorization and Sanitization should be in a different layer
  - Isolation of generated code is highly recommended
  - The best for user personal features: Generated code is running public SDK commands, executed on isolated environments.



Flaky software Testing

- While testing can be automated as well, the basic dev flows are challenged.
- Once LLMs are part of the app tests are becoming hard:
  - The “standard flows” are dependent on LLM response, often nondeterministic
  - Tests are slow, unstable and flaky
  - Caching the responses eases the pain but... hides many issues along the way
- Additional testing phase is needed, AI Testing:
  - Basic tests contain only:
    - Super simple, guaranteed to work cases.
    - Cached LLM responses.
  - AI Testing phase:
    - Test the LLM interactions parts
    - Very similar ML evaluation, with some extra layers.

Evaluation is extremely hard

- LLMs in general are very hard to evaluate, a process often done manually in low volume
- The only way to evaluate our code (which is increasingly becoming LLM generated) :
  - Strong focus to PRD, Arch and design:
    - Entities are well defined
    - Every interaction is an API
    - Test, Many automatically generated tests.
- Don’t try to validate the output, Focus on putting strong guard rails.
- These guard rails mean A LOT OF TESTS, no worry, AI can help here as well.

Words matter (AKA Langware☺)

- Architecture, Specs, PRDs are becoming more important then ever
- Words are becoming the foundation of software development
- Every error is greatly amplified by automation
- The documents are the product, hence past acceptable problems are becoming automation blockers:
  - Outdated specs
  - Undefined requirements
  - Incorrect documented facts (tickets)
  - Lack of solid architecture
- Expect significant transformation of the product roles, tools and methodologies.