

Generative AI assisted development

- Practical experiences

FIIF seminar 10.10.2024

Harnessing the power of LLMs like
ChatGPT in software development

Antti Lammi

futurice



WE ARE FUTURICE

Nordic roots, global mindset

PEOPLE
800+

COUNTRIES

6

NATIONALITIES

63

CONS. YEARS OF GROWTH

23

Care. Trust. Transparency.
Continuous improvement.

These are our core values, and the cornerstones of our company culture. They define how we work, provide continuity in a changing world, and keep us unique. The worth of our culture is determined by what happens when nobody is watching.

Portugal

London

Stuttgart

Berlin

Munich

Poland

Stockholm

Helsinki

Tampere

FAMILY OF COMPANIES



Digital commerce consultancy

meltlake°

A Microsoft specialist consultancy

THRIV

Hand-picked developers with passion and expertise

RECORDLY

Future-proof, robust, and secure data solutions



Senior Lead developers, architects and product managers



Interim and fractional executive services

QLARIFY

Quality engineering, coaching, and leadership.

Today: Developer Point of View

Coding and Development

- Code Generation
- Code Review and Quality Assurance
- Pair Programming
- Documentation
- Prototyping
- Code conversion
- SQL fine-tuning
- ...

Requirements Gathering and Analysis

- Natural Language Processing (NLP)
- Predictive Analysis

Design

- Automated Design Prototyping
- UI/UX Optimization

Deployment

- Predictive Deployment
- Continuous Integration/Continuous Deployment (CI/CD)

User Feedback and Adaptation

- Sentiment Analysis
- Feature Usage Analysis

Quality Assurance

- Automated Test Case Generation
- Defect Prediction

Maintenance and Operations

- Predictive Maintenance
- Log Analysis



BLACK

WHITE

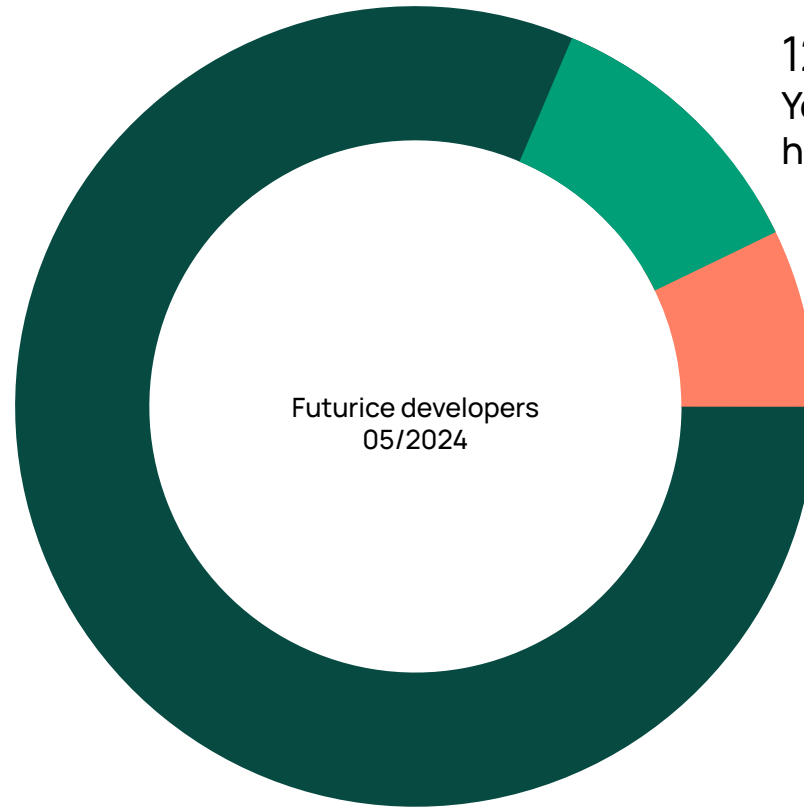
Survey for Futurice Developers

We conducted an exploratory survey targeting our own developer community to gain insights into the current landscape of AI-powered coding tools. The survey aimed to understand the adoption, use cases, perceived productivity impact, effectiveness, challenges, and overall sentiment towards generative AI tools within our development processes.



Are you using Gen AI tools in software development?

82%
Yes, including
client projects



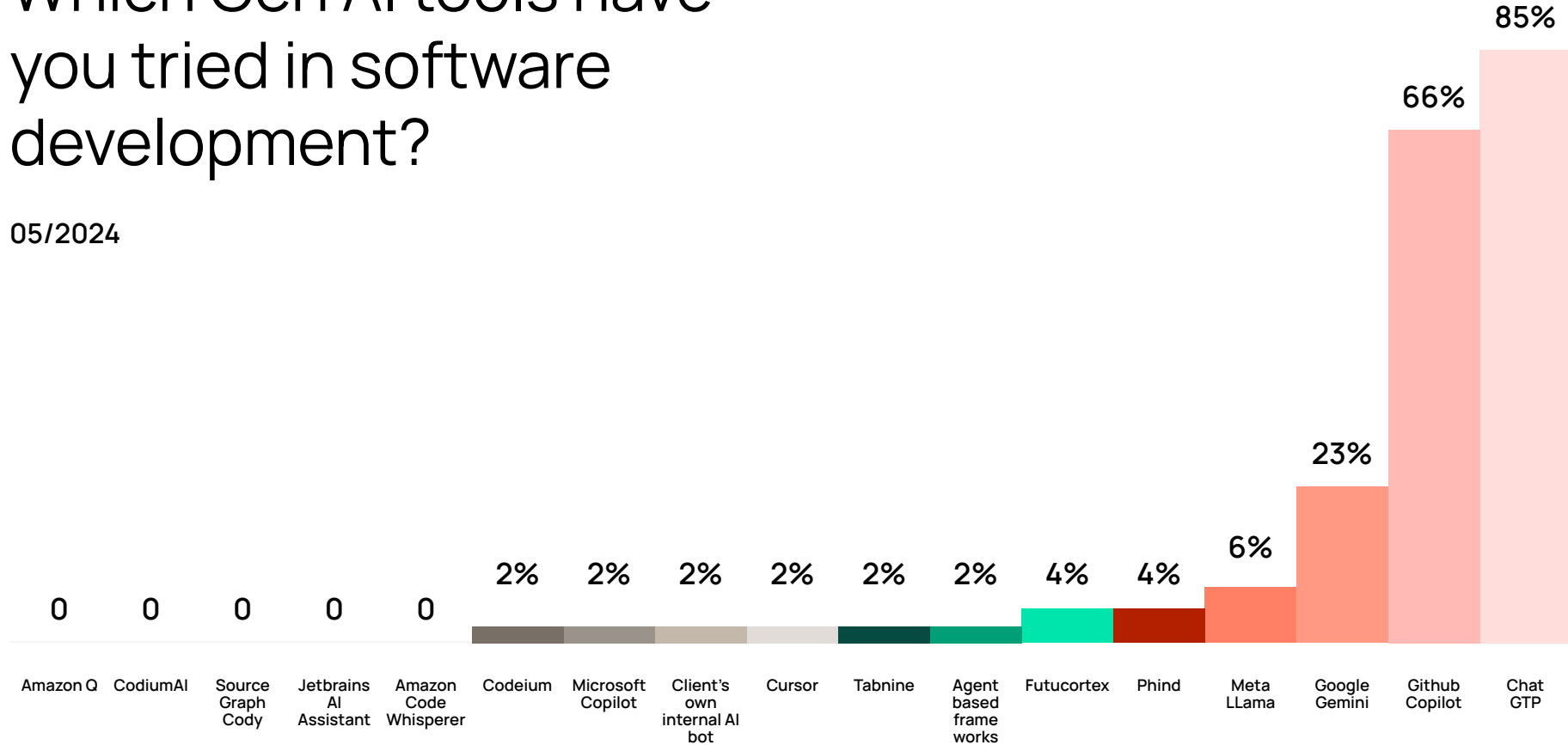
12%
Yes, but only in
hobby projects

6%
No

Futurice developers
05/2024

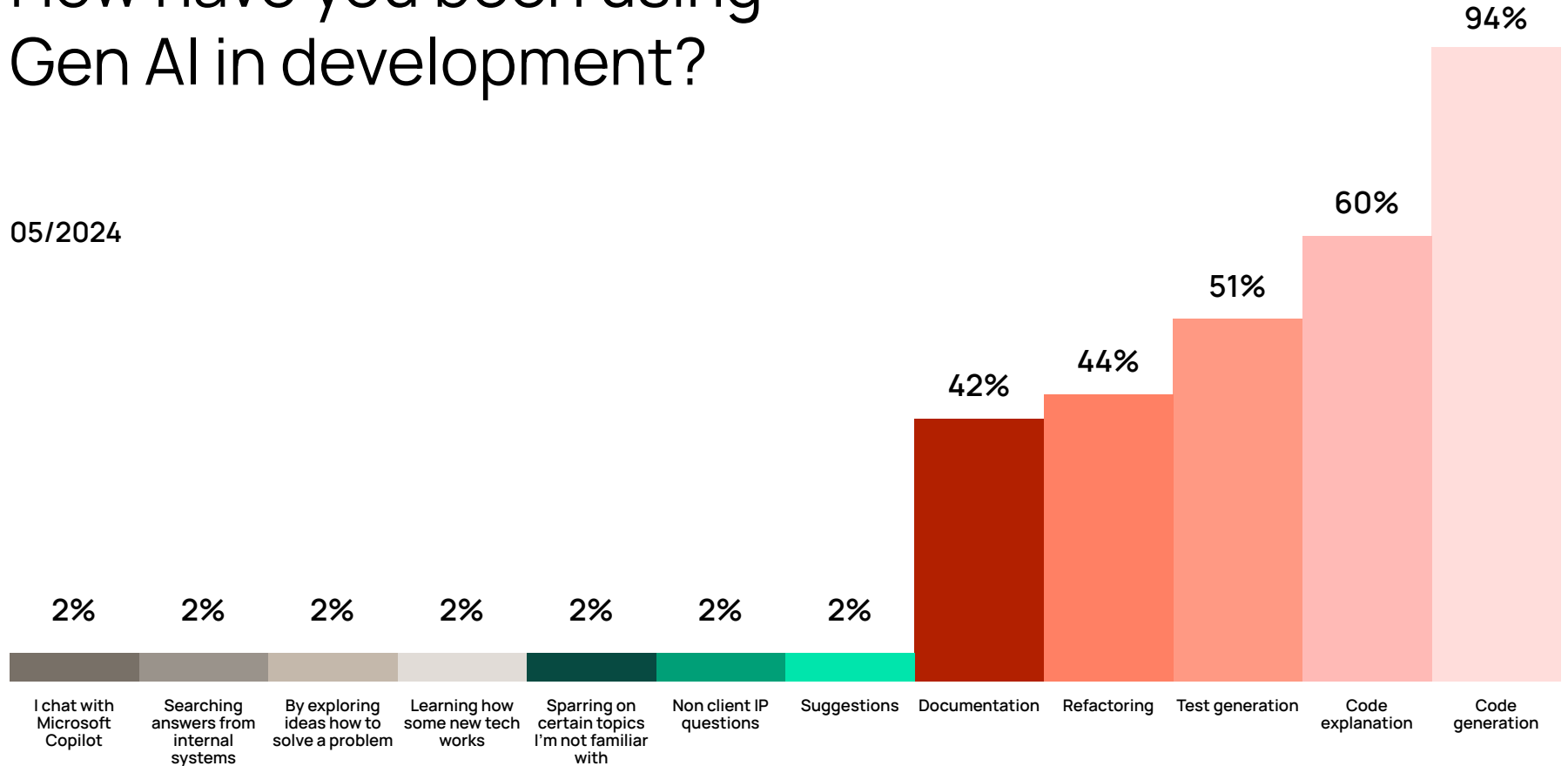
Which Gen AI tools have you tried in software development?

05/2024



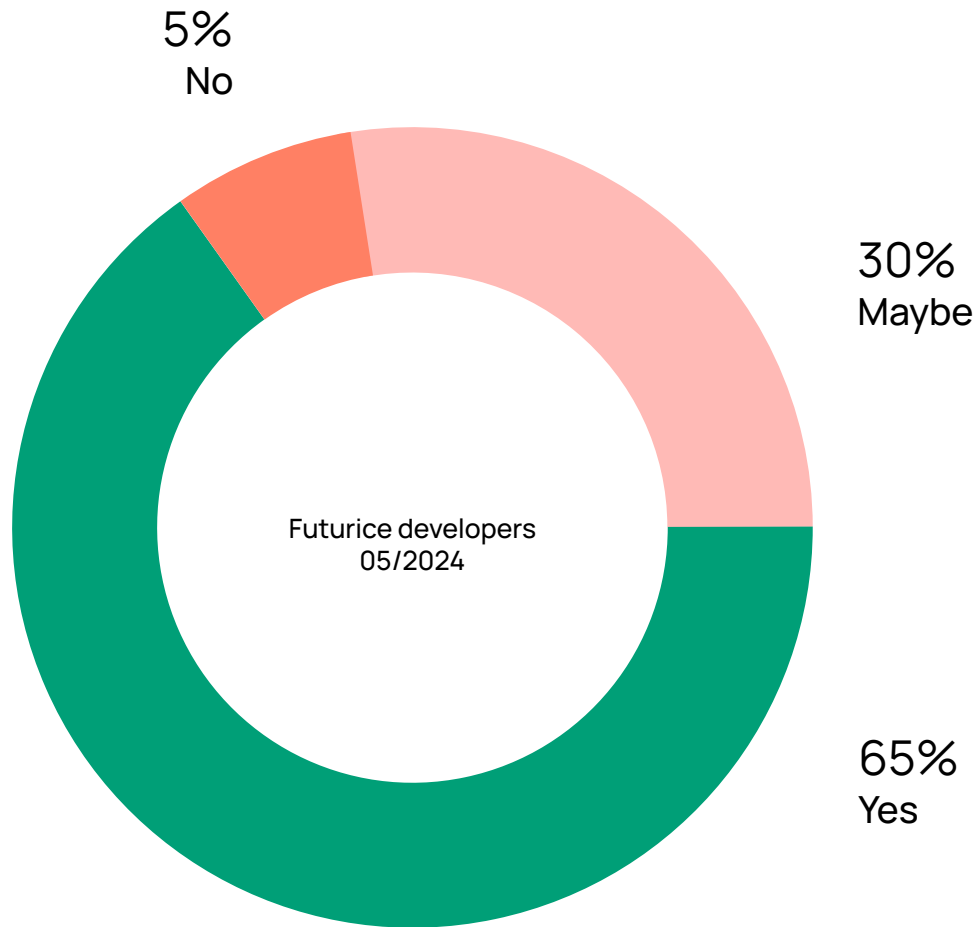
How have you been using Gen AI in development?

05/2024



Do you think these tools make you more productive?

05/2024



Features and their usefulness

From the coding perspective

Offered features can be roughly divided to the following most useful ones. Subjective ranking and opinion on usefulness.

Autocomplete

Purpose

Autocomplete on word / line level. Competing with your IDEs normal autocomplete.

Effectiveness

Superb! Gives a noticeable productivity boost. Able to autocomplete stuff your standard IDE/Language server cannot.

Pattern generation

Purpose

Autocomplete larger chunk of code based on already existing code. Cases: pattern matching, repetitive code (case study later), missing if branches, adding more test cases following the same test pattern.

Effectiveness

Superb! Can save a lot of manual typing. Sometimes really excels on creating test case boilerplate code.

Implement

Purpose

Implement small code snippets or functions with clearly defined input/output and definition. Most effective if no dependencies to anything other in workspace. Not for feature implementation.

Effectiveness

Mediocre. Hallucinations start to occur. Works 50/50.

Explain

Purpose

Explain some piece of code.

Effectiveness

OK. Can give good results and sometimes be completely off.

Documentation

Purpose

Write basic docs on a function level.

Effectiveness

Quite good to state the obvious but in cases where there really is a need for code comments we have not had much success.

Fix / Refactor

Purpose

Work on a piece of code that has some issue or that would use some refactoring.

Effectiveness

Bad. Often ends up taking more time than doing it yourself.

Generate tests

Purpose

Generate test cases from scratch. Only applicable for testing those simple well defined functions. In a real world project with many dependencies and often related mocked modules this does not work.

Effectiveness

Bad. Can create tests mostly for things not worth testing (depends on your testing philosophy).

Case study

Autocomplete: Pattern generation

1 Set the scene

We need to apply a given pattern for a set of values, Here we have provided a pattern but still need to add more values for the message status.

TS generation.ts ●

TS generation.ts > ...

```
1
2   export const MESSAGE_STATUS_PENDING: MESSAGE_STATUS = 'pending';
3   export type MESSAGE_STATUS = 'pending';
4
5
6
7
8
9
```

2 Generate

Using the copilot inline chat we get a great success for filling in the values.

TS generation.ts ●

TS generation.ts > ...

1

add statuses DRAFT, PUBLISHED, DELETED, ARCHIVED

GitHub Copilot

Made changes.

Accept

Discard



Accept or Discard 1 change.

```
2 export type MESSAGE_STATUS = 'pending' | 'draft' | 'published' | 'deleted' | 'archived';
3 export const MESSAGE_STATUS_PENDING: MESSAGE_STATUS = 'pending';
4 export const MESSAGE_STATUS_DRAFT: MESSAGE_STATUS = 'draft';
5 export const MESSAGE_STATUS_PUBLISHED: MESSAGE_STATUS = 'published';
6 export const MESSAGE_STATUS_DELETED: MESSAGE_STATUS = 'deleted';
7 export const MESSAGE_STATUS_ARCHIVED: MESSAGE_STATUS = 'archived';
```

8

9

Case study

Working with unknown languages or frameworks

1 Building a project

When kickstarting new projects, we draw on what we have experienced before.

Conceptually GenAI has seen many projects and can summarize what it has seen.


Keeping the questions simple, it is possible to drill into the **how & why** of starting project.

Combine this with your own human experience on what *really* matters when delivering projects.


Think of it as the perfect blog post you find where somebody has written about doing 90% of what you want to do (but with interactivity!)

CHAT



 grahamdaw

Give me an example project structure for a FastAPI project

 GitHub Copilot

A typical FastAPI project might have a structure like this:

```
/myproject
/app
  __init__.py
  main.py
  /routers
    __init__.py
    users.py
    items.py
  /models
    __init__.py
    users.py
    items.py
  /schemas
    __init__.py
    users.py
    items.py
/tests
  test_main.py
  test_users.py
  test_items.py
.env
.gitignore
Dockerfile
requirements.txt
README.md
```

Here's what each part does:

- `main.py`: This is where you create your FastAPI application, register your routers, and configure any middleware.
- `/routers`: This directory contains different routers (similar to controllers in other frameworks). Each router is responsible for a specific part of your API (e.g., users, items).
- `/models`: This directory contains Pydantic models, which you use to validate the data your API works with.
- `/schemas`: This directory contains Pydantic schemas, which you use to structure the data your API returns.

2

Augmenting knowledge with LLMs

“How create a decorator in FastAPI?”

“In FastAPI how to get request headers?”

“What does the `model.dumps()` do?”

Each time suggestions would come - sometimes good, sometimes bad.

But at the core it replaced what would take copious amounts of searching, experimenting and documentation with very targeted suggestions and solutions.

Keeping clear context, simple questions and, most of all, iterating the results will get much better results.



grahamdaw

@workspace /explain Write an explanation for the active selection as paragraphs of text.

```
@contextmanager
def http_exception_handling():!
```

GitHub Copilot used @workspace /explain

Used 1 reference >

The active selection is a Python function declaration that is intended to be used as a context manager for handling HTTP exceptions.

The `@contextmanager` is a decorator provided by the `contextlib` module in Python's standard library. This decorator is a way to define a factory function for `with` statement context managers, without needing to create a class or separate `__enter__()` and `__exit__()` methods.

3 Collaboration not automation

GenAI is not a silver bullet.

"Make me a FastAPI API that has these endpoints that integrate to this and considers that..."

The problem is currently too complex and the context required is too large.

These current generation tools are not going to automate away our need to understand and write code.

However, these developer tools generally produce good results with simple contexts and clear objectives.

These tools are for collaboration where humans and GenAI build solutions together.

GenAI tools used



<https://github.com/features/copilot>



<https://chat.openai.com/chat>

phind

From idea to product

<https://www.phind.com/>



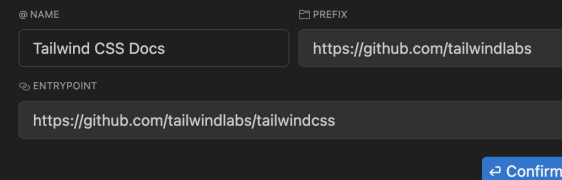
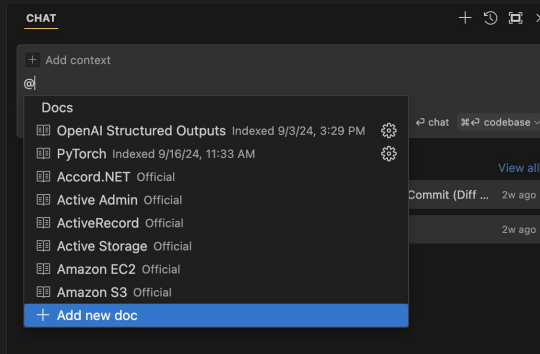
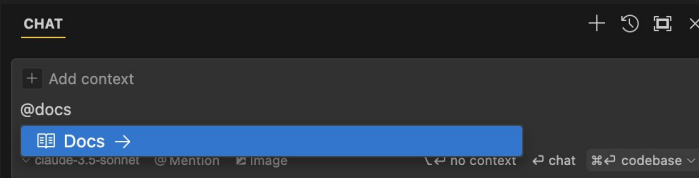
<https://www.cursor.com/>

Case study

Cursor - VS Code Replica with AI Native Features

1 Adding documentation to your question

You can give a URL that Cursor will then “eat” - that documentation then becomes “taggable” when writing new prompts.



Make sure the component follows the documentation from @Tailwind CSS Docs

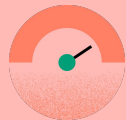
claude-3.5-sonnet @ Mention Image no context chat codebase

Learnings and considerations

BERLIN / HELSINKI / LONDON / MUNICH / STOCKHOLM / STUTTGART / TAMPERE

futurice

What have we learned?



Speed of tool development is blinding

What we learn today might be obsolete next month, week, day.

Cursor type intelligent agent seems to be the current spearhead



**Good:
Learning new
Power-Google
Autocomplete**

Invaluable tool when you are not a deep expert on the chosen tech stack

Why Google / Stack Overflow anymore?

Autocomplete is magic & a source of frustration



**Bad:
Complexity
Validation
Hallucinations**

GenAI cannot currently handle complexity and dependencies

Would not suggest using code without human validation

Hallucinations from thin air can cause serious negative attitude towards the tools



Starting development with GenAI in mind

Almost all of our current experience is with using GenAI in existing projects

Development nor tools are not geared towards machine-produced code

Context is king

IDE integrated tools only as good as their ability to figure out the appropriate context.

Going deeper, GitHub Copilot is powered by OpenAI Codex. The auto-generated suggestions come from the context within the file, like function names, code comments, docstrings, file names, cursor position, and more.

Figuring out the proper context is likely going to be the differentiating factor between the tools.

Always remember that Generative AI powered tools, especially related to developer assistance, will be more likely to produce good results when there is:

- **Just the right amount of context**
- **A clear objective**



Thank you

futurice





futurice

Empowering the world to act.