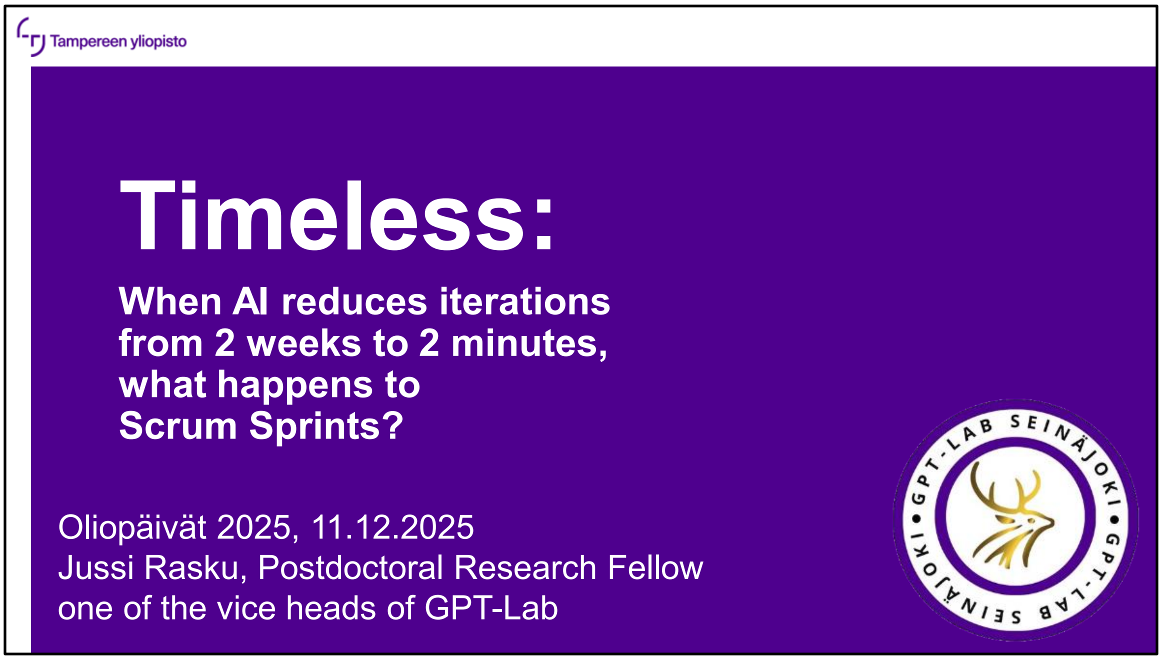


Tampereen yliopisto

Timeless:

When AI reduces iterations from 2 weeks to 2 minutes, what happens to Scrum Sprints?

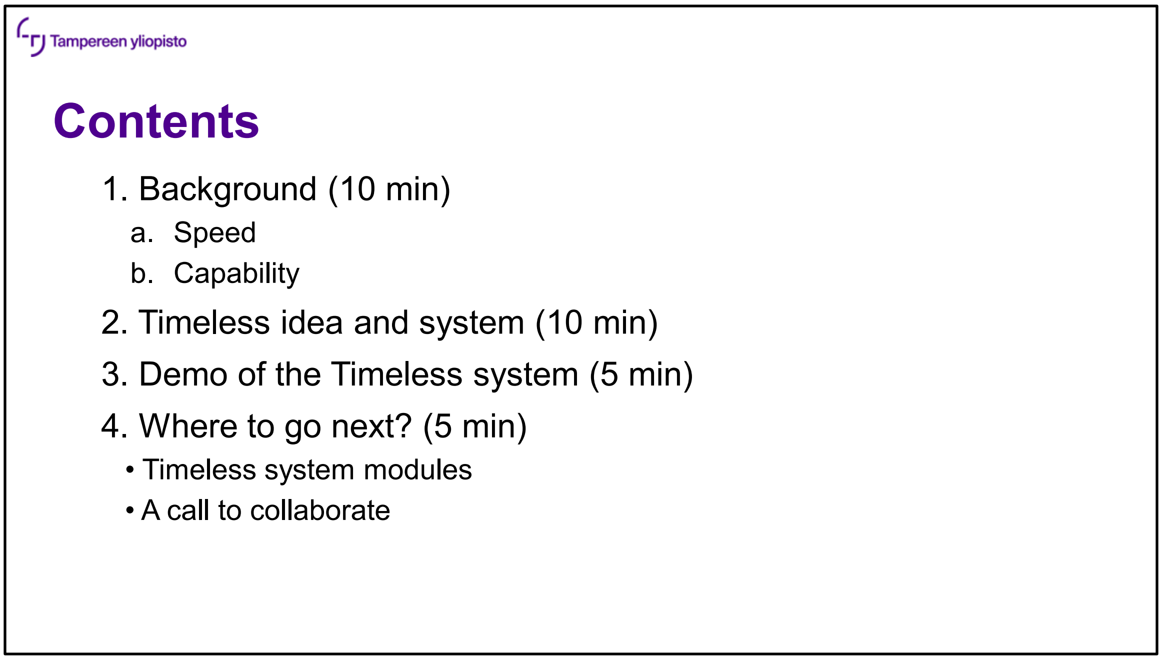
Oliopäivät 2025, 11.12.2025
Jussi Rasku, Postdoctoral Research Fellow
one of the vice heads of GPT-Lab

GPT-LAB SEINÄJOKI • GPT-LAB SEINÄJOKI •

Jussi, introduce yourself.

Theme/idea: the field is moving on a breakneck speed. This means that we need to. We at GPT-Lab have been asking: If 2 week sprints become 2 minute sprints, what impact does that have on how we create software?

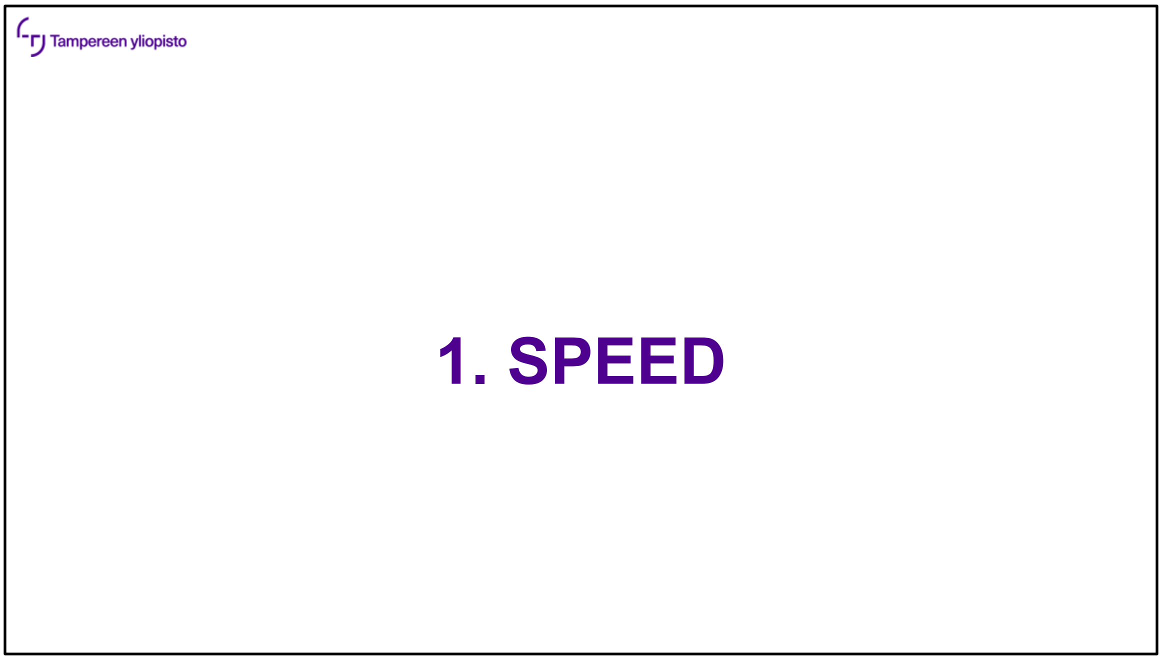
Let’s explore it together!


Tampereen yliopisto

Contents

1. Background (10 min)
   a. Speed
   b. Capability
2. Timeless idea and system (10 min)
3. Demo of the Timeless system (5 min)
4. Where to go next? (5 min)
   - Timeless system modules
   - A call to collaborate

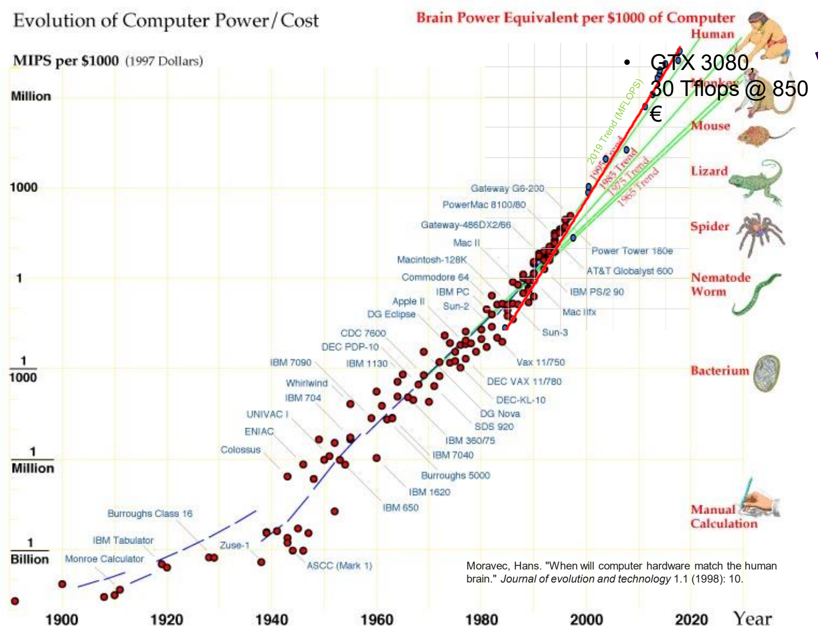
I have a hopeless tendency for time optimism (all takes less time in my head than it actually does), so I hope we can discuss all this.
At least we need to storm through the background to have time to discuss the Timeless idea and show the operational system.



1. SPEED

The first important aspect of AI in software engineering I want to discuss with you is the time dimension. The speed.

Why now?



- GTX 3080, 30 Tflops @ 850 €

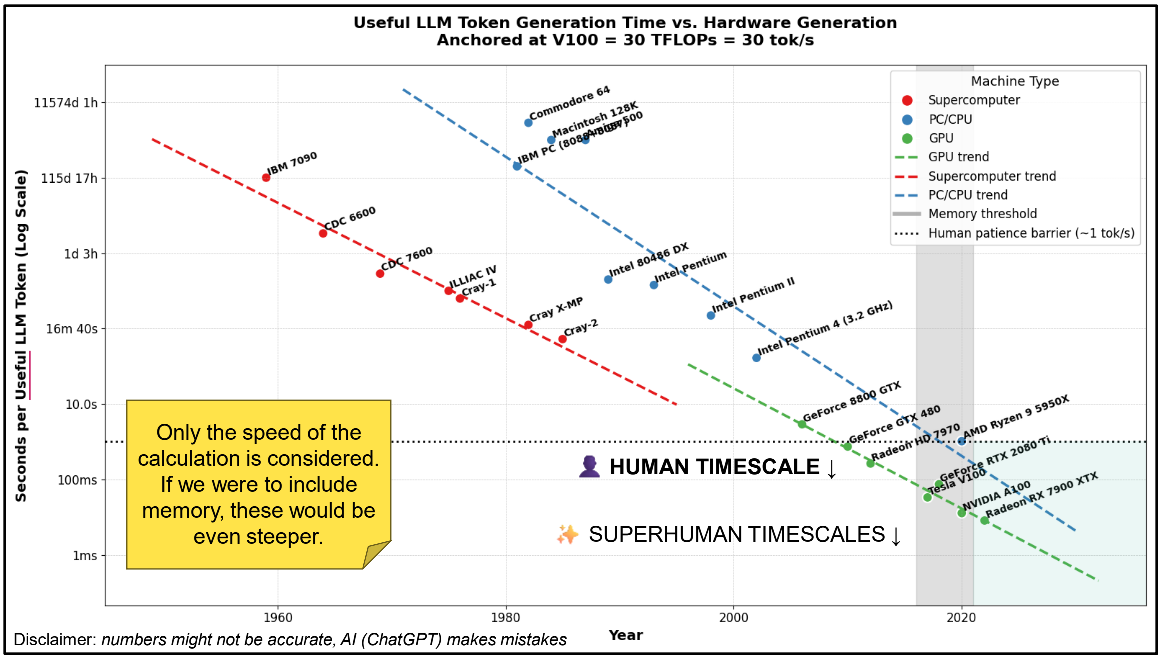
There seems not be technological barriers to stop this.

We need to first acknowledge the huge increase in computing power.

Why now? Why do we only now have AI that can solve coding exercises and demonstrate reasoning abilities that are close to or at times even surpass human capabilities? This graph from 1998 has great explanatory power: it tells how much computing power you can buy with 1000 $/€. Most of us started in an era of computers with nematode's worth of computing power. Nematode can't integrate, no matter how much you train it. Nor a lizard can’t write you an implementation of an online store. It was only when we reached a level where a computer has roughly mid-sized mammals worth of computing that we have built programs that think and can make honest attempt in solving any problems.

The scientist should mention that there are other kinds of estimates, mainly larger, of the brain's computing capacity. However, these estimates do not change the exponential (and see the trendlines, accelerating) increase in computing power. https://aiimpacts.org/brain-performance-in-flops/

Unless we hit a barrier, and there seems not to be one, there will be interesting times ahead.



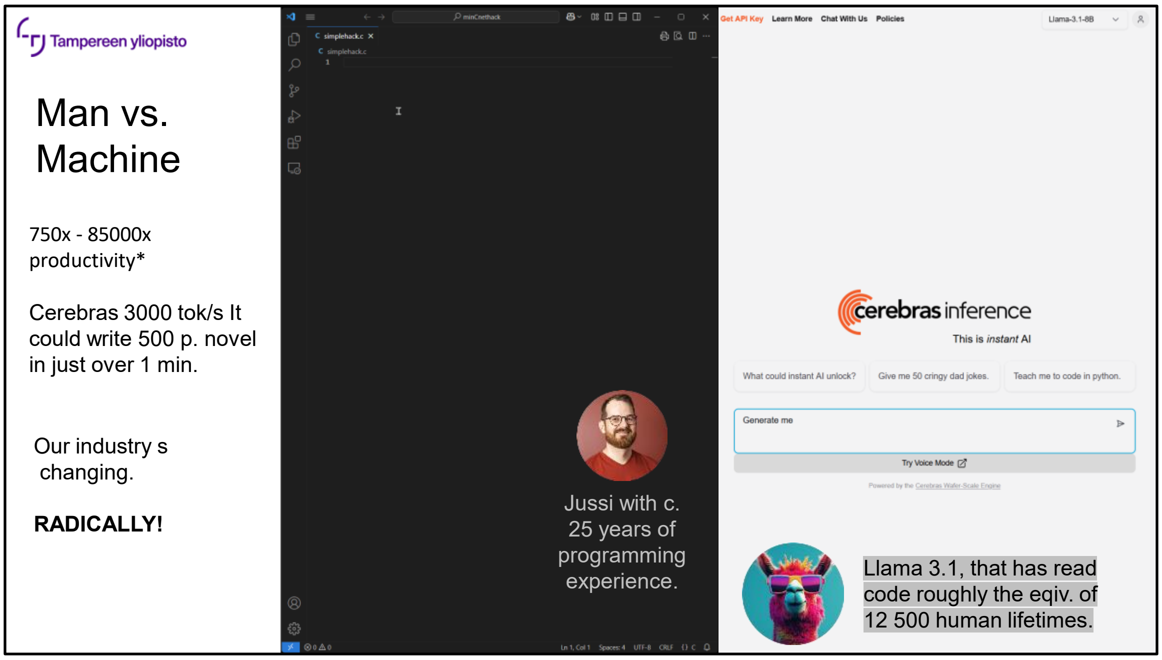
Similar information from a slightly different perspective: How much it takes (and would have taken) for the computer emit one token.

This plot also tries to explain why large-scale LLMs like GPT-3 weren't feasible before ~2020: computers just weren't fast enough, and the memory capacity just wasn't there. Only recently did we exceed the thresholds where:
1. Token creation is fast enough for practice/reasoning
2. Useful size models (say 13B) can fit weights in memory

Only when these both preconditions become reality, and this is the key point, will interactive applications (chat, code, search) be usable for an impatient person. LLMs are the result of hardware curves crossing key usability thresholds – not just to get better ideas.

And the future: Even greater speeds.

Details: E.g., for GTX 480 ~1–2 tok/s for FP16/FP32-like paths, ~3–5 tok/s for aggressive quanta (Q4_K_M).



Man vs. Machine

750x - 85000x productivity*

Cerebras 3000 tok/s It could write 500 p. novel in just over 1 min.

Our industry s changing.

RADICALLY!

Jussi with c. 25 years of programming experience.

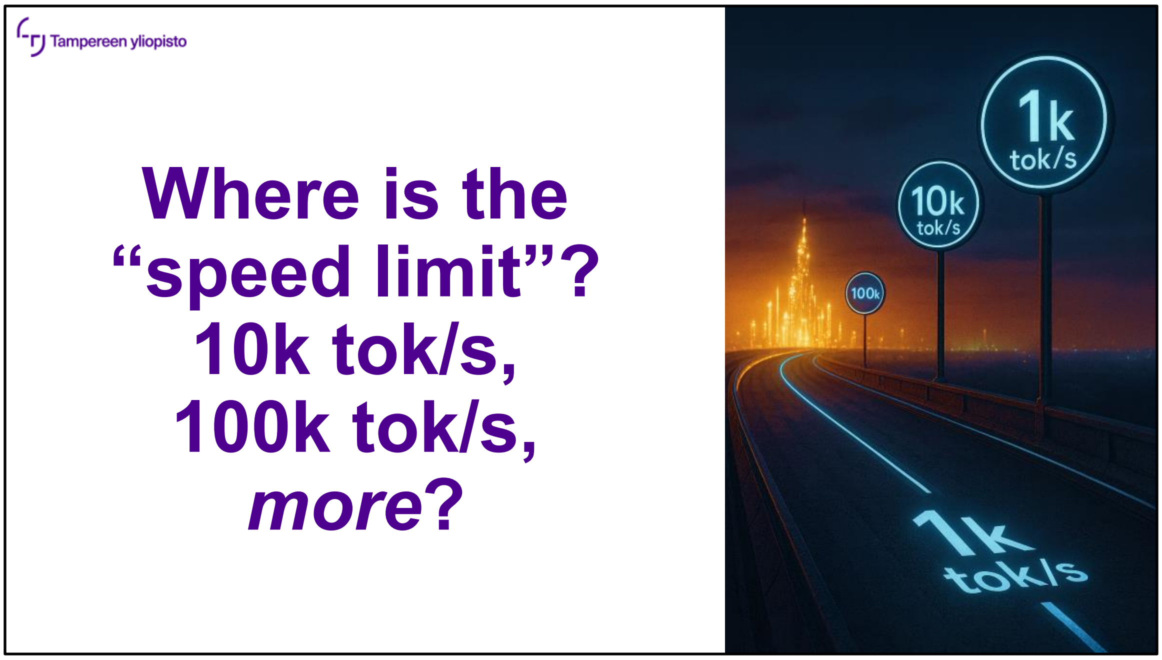
Llama 3.1, that has read code roughly the eqiv. of 12 500 human lifetimes.

Q:How much faster these systems are than us? A:Many, many times.

The implications of a speed over 2000 WPS are difficult for a person to grasp. That is already roughly 10x the speed the fastest typists (200 WPS) can write. And that also includes some artificial "thinking" on producing output that actually considers the earlier text and problem definition.

*) On average, a programmer creates 5 or 15 or 150 lines of code per day (who knows for sure, see DoD Software Factbook v1.1 , Cusumano et al. (2003) and others) so with 3000 tok/s and 10 tok/line for a 24h day (vs 8h) we are already at 86000x through 30000x to 3000x. Even if we try to take into account everything that a software developer does – meetings, documentation, instant messaging, emails, etc. (trying to estimate the volume of this in tokens) – the multiplier is still ~750x (have lost my notes on this, will revisit).

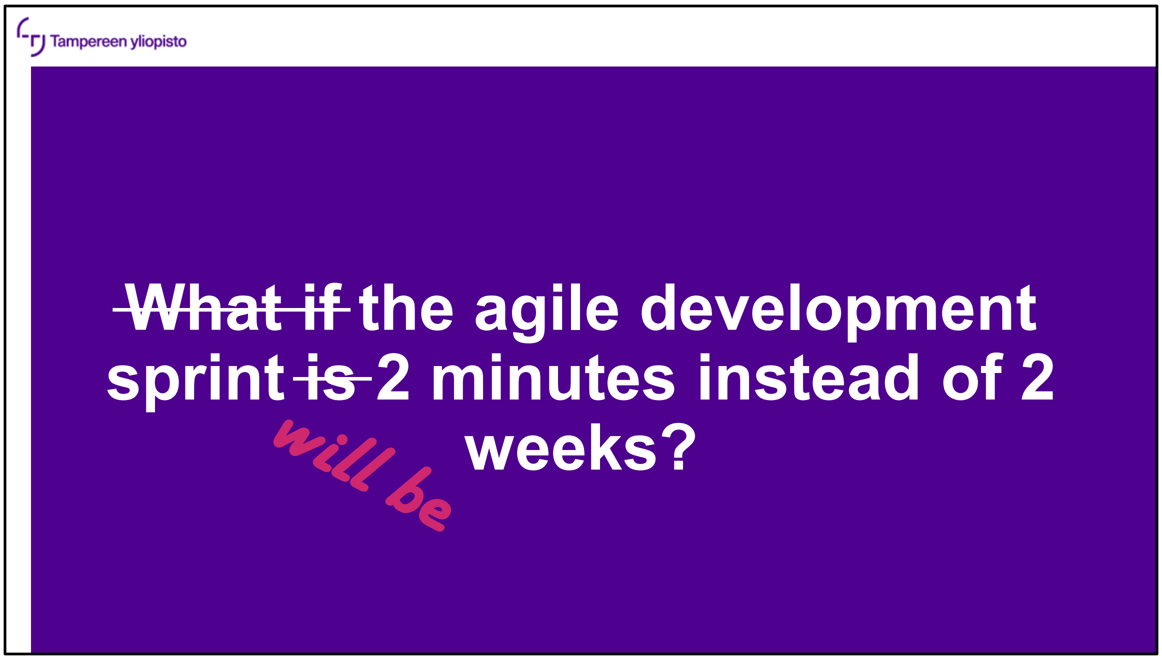
For example, a 500-page novel that is approximately 170,000 words would take the Cerebras service 1 minute to write. Start to finish. Would that be good? Maybe not yet, taking into account the speed of development of artificial intelligence. What about in a year's time? After five years?



Tampereen yliopisto

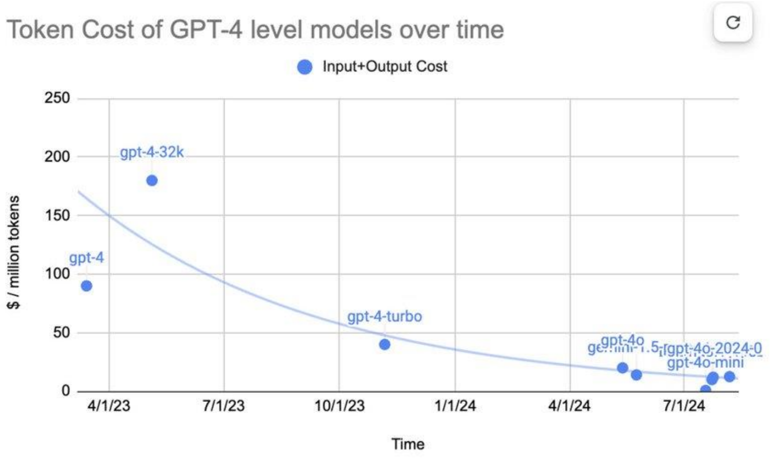
Where is the "speed limit"? 10k tok/s, 100k tok/s, *more*?

Based on the earlier charts, the growth in computing capability just seems to continue, or even accelerate. We have interesting times are ahead. We already have superhuman content generation speed, and this seems to continue to even more hard to comprehend speeds in artificial cognition.


Tampereen yliopisto

~~What if~~ the agile development sprint ~~is~~ will be 2 minutes instead of 2 weeks?

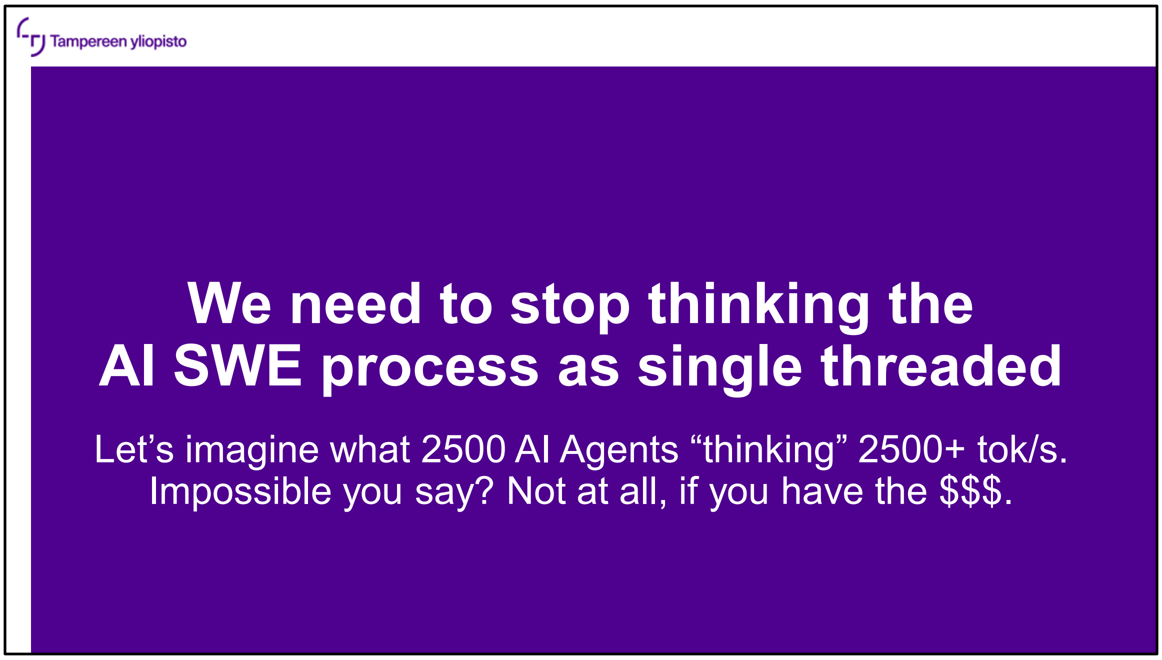
Tampereen yliopisto



Cost for 2 million tokens (input+output) decreased from $180->$0.75 in 2 years. 240x cheaper

Source: [Elad Gil on X]

Similarly, as we need to consider time, we need to consider cost: AI inference cost is collapsing toward negligible. Competition and open-source supply are forcing rapid price convergence. Hardware and software optimizations cut $/token and raise throughput. On-device models push cost toward zero for common workloads. Core implication: compute-rich reasoning becomes more and more affordable.



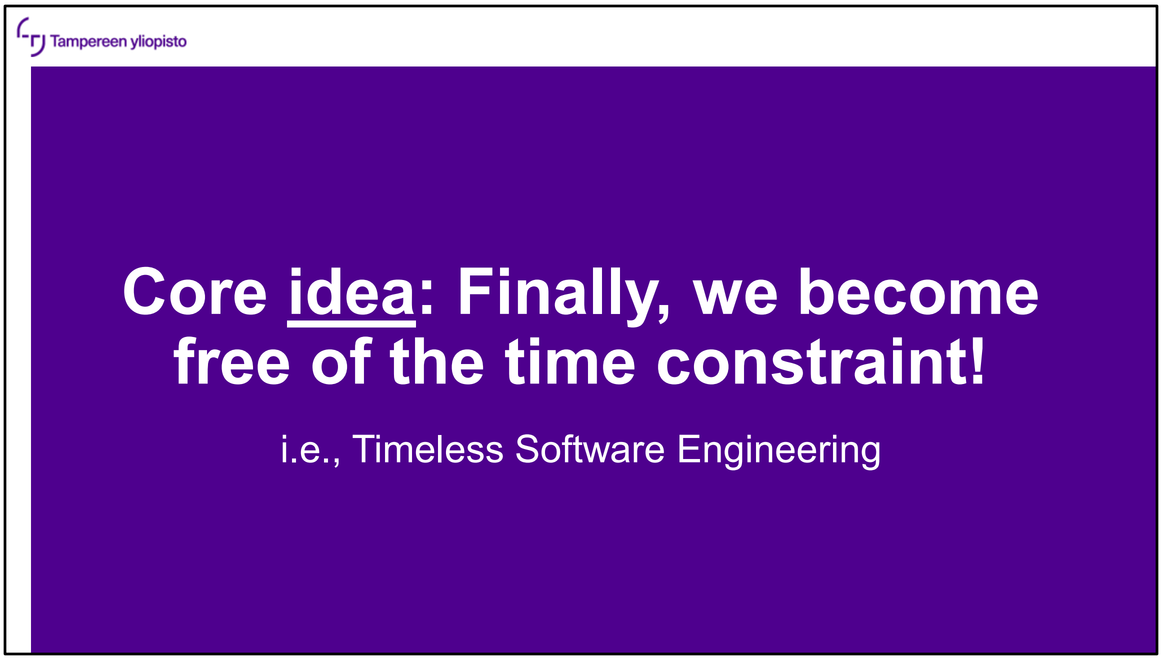
Tampereen yliopisto

We need to stop thinking the AI SWE process as single threaded

Let’s imagine what 2500 AI Agents “thinking” 2500+ tok/s.
Impossible you say? Not at all, if you have the $$$.

Ensembles have proven out to be one of those surprisingly powerful ideas. Strategic move: **parallelize artificial cognition**—run many reasoning paths at once instead of deep sequential chains. **Build for a world where parallel cognitive bandwidth matters more than token cost.**

On the example: running such a system would cost thousands of dollars per hour, but in a couple of years it could be hundreds.
The thought experiment involves the system producing usable results.

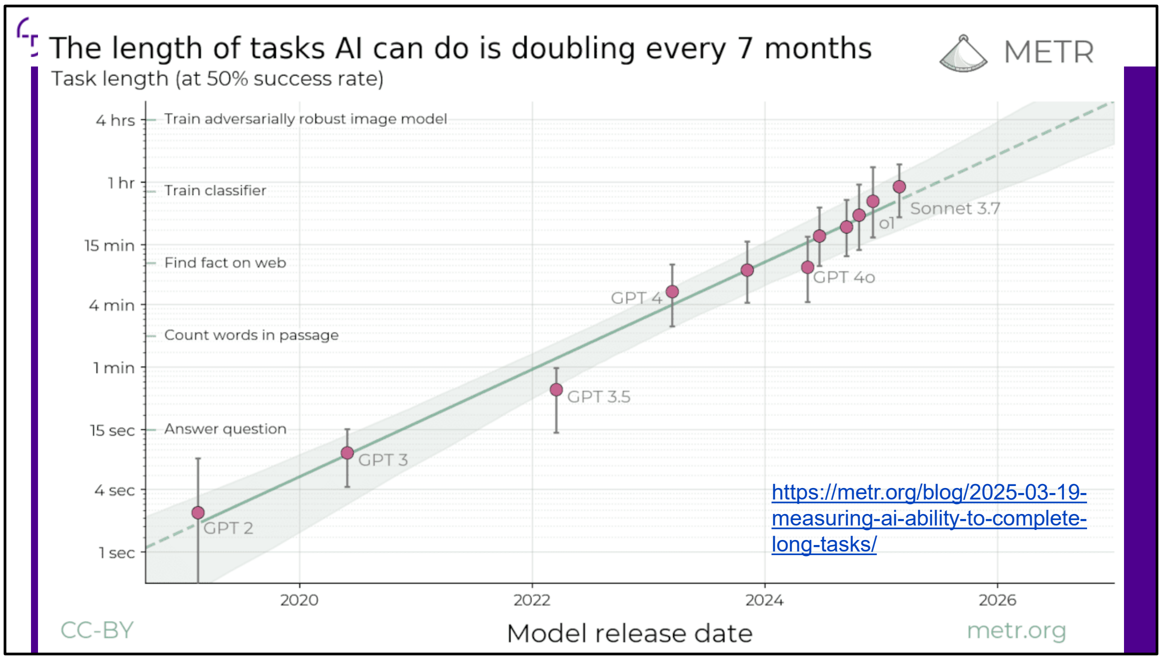


Tampereen yliopisto

Core idea: Finally, we become free of the time constraint!

i.e., Timeless Software Engineering

There are so many things we cannot do b/c we do not have the time. The superhuman speed and parallel scalability of AI can help.

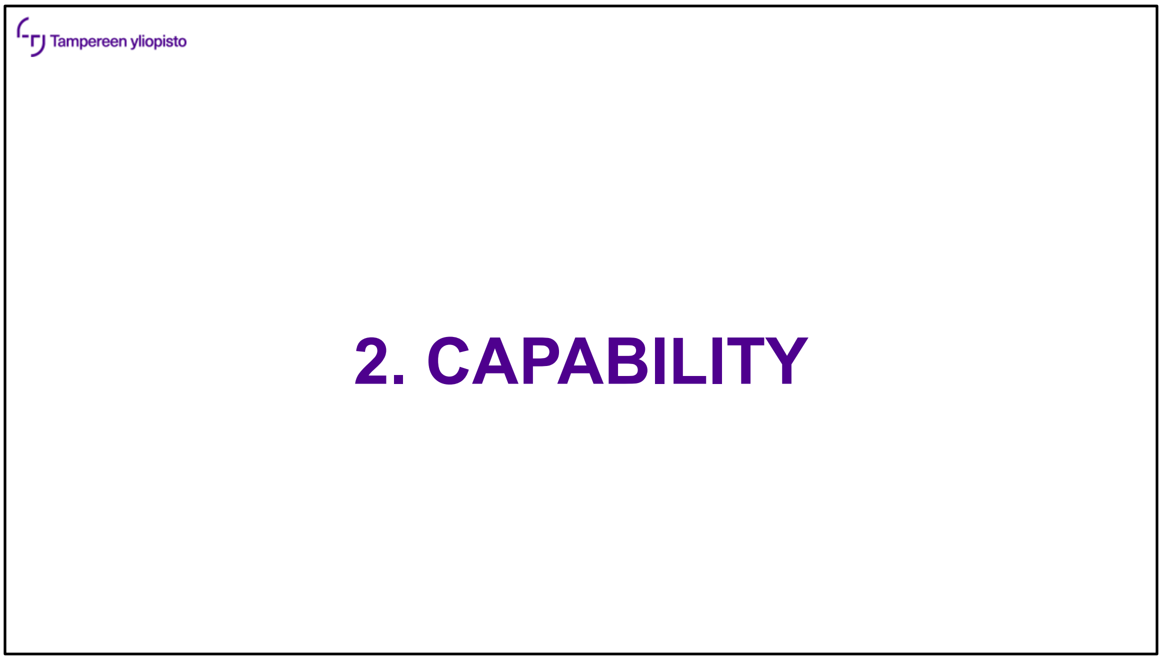


https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/

The increase in speed is indirectly inked to capability.

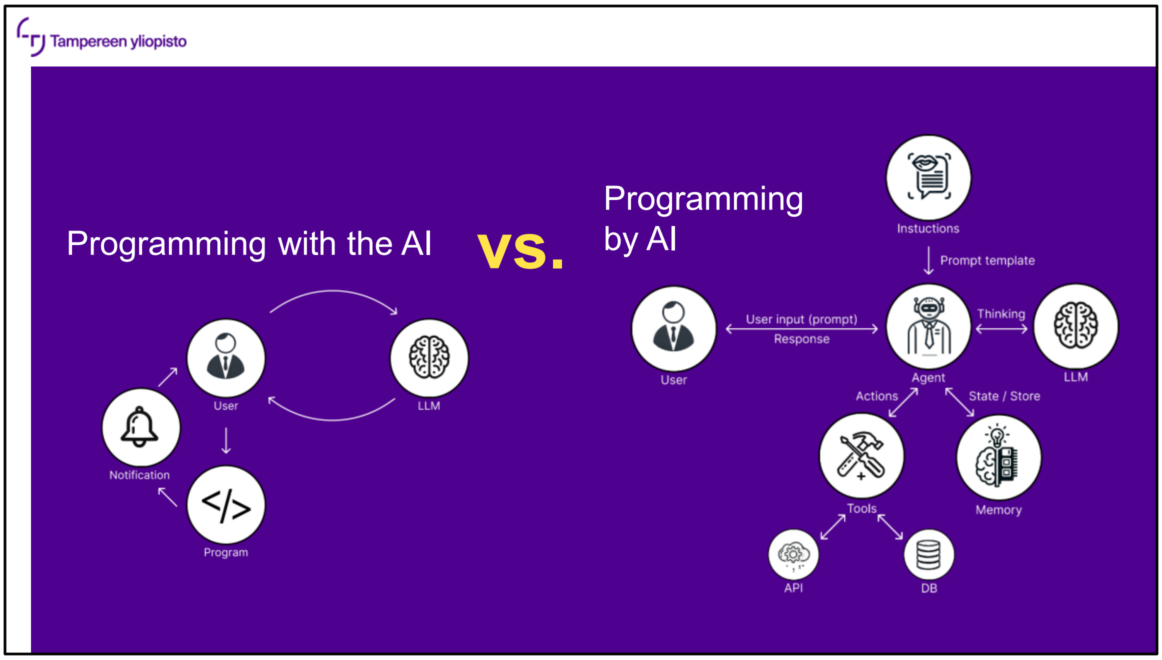
For example, according to the data gathered by METR (Model Evaluation & Threat Research nonprofit), there is a clear trend in how frontier models can solver bigger and bigger tasks. The length of task (when measured in human minutes) the AI models are able to solve is doubling every 7 months.

But , and there is but: As keen eyed already spotted, the 50 % success rate in this chart is not that great.


Tampereen yliopisto

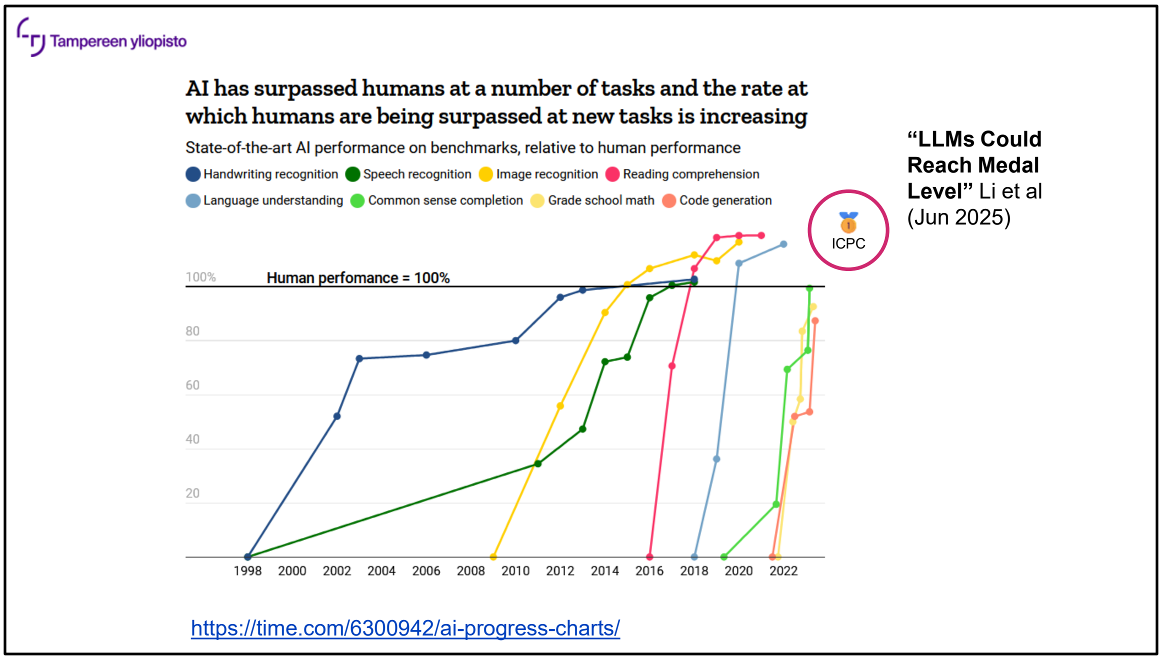
2. CAPABILITY

And this is why we must also discuss capability next.



Because of the rapid increase in capability, we have already been moving away from the programming with AI to a world where programming is done by the AI and agentic systems.

We are building agentic systems that have more autonomy, memory, can use tools, have specific role and purpose to reach its goals.


Tampereen yliopisto

AI has surpassed humans at a number of tasks and the rate at which humans are being surpassed at new tasks is increasing

State-of-the-art AI performance on benchmarks, relative to human performance

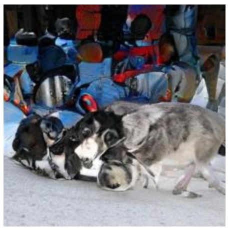
"LLMs Could Reach Medal Level" Li et al (Jun 2025)

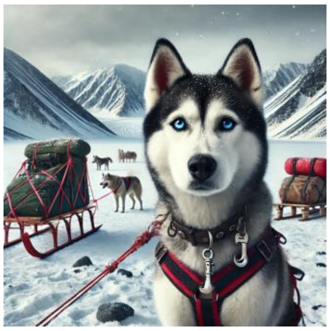
https://time.com/6300942/ai-progress-charts/

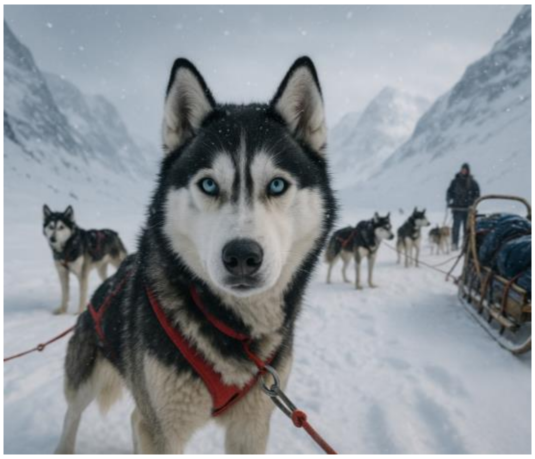
Due to the previously mentioned development of computing power, people are losing to machines one after the other in tasks that require cognitive abilities.

There are already many tasks where a machine beats a trained human, and as AI capabilities grow, it is only a matter of time before the human level is reached and exceeded in others as well. This also applies to software engineering tasks.
--
Note: The data that is a couple of years old. According to a study published in June 2025, advanced inferential models are already capable of medal-level performances in ICPC competitions at International Collegiate Programming Contest Finals Medal level: https://arxiv.org/html/2506.12713v1 (Humanity's Last Code Exam**)**

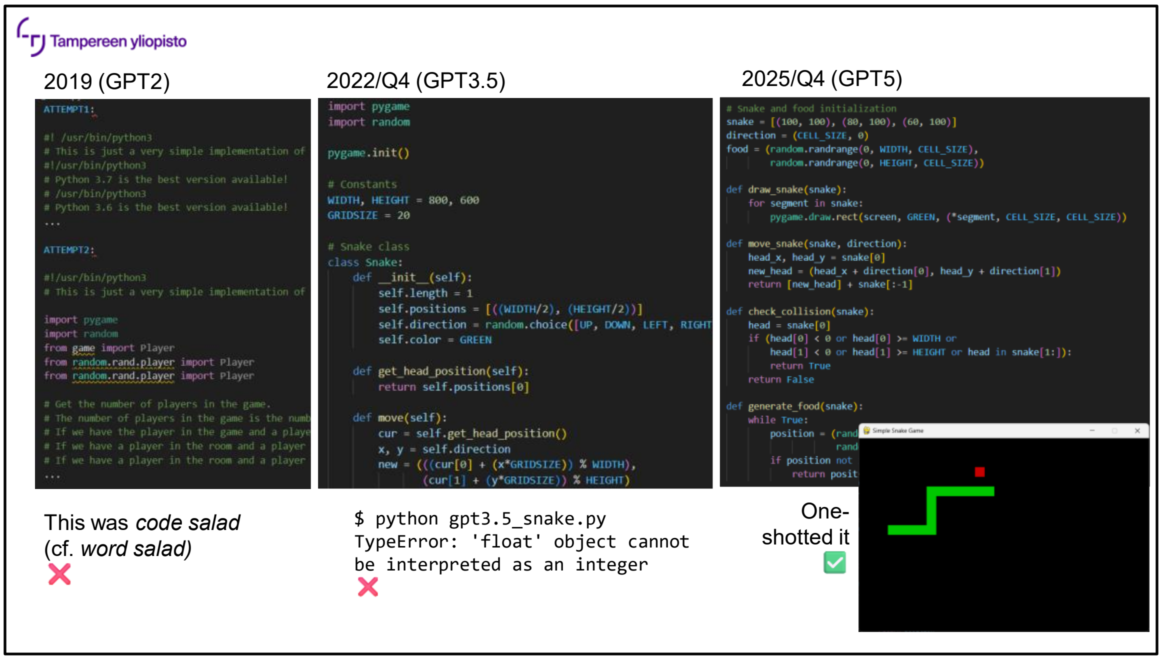
Tampereen yliopisto


2020


2024


2025

You get used to the amazing achievements of artificial intelligence amazingly quickly. However, all you have to do is look back and see that a huge development has taken place. Here's a picture from five years ago (something like fur?), and a year ago, as well as the latest model. It's amazing how quickly our expectations are changing, and magical technology is becoming "normal."



Tampereen yliopisto

2019 (GPT2)

```
ATTEMPT1:

#! /usr/bin/python3
# This is just a very simple implementation of
#!/usr/bin/python3
# Python 3.7 is the best version available!
# /usr/bin/python3
# Python 3.6 is the best version available!
...

ATTEMPT2:

#!/usr/bin/python3
# This is just a very simple implementation of

import pygame
import random
from game import Player
from random.rand.player import Player
from random.rand.player import Player

# Get the number of players in the game.
# The number of players in the game is the numb
# If we have the player in the game and a playe
# If we have a player in the room and a player
# If we have a player in the room and a player
...
```

This was *code salad*
(cf. *word salad)*
❌

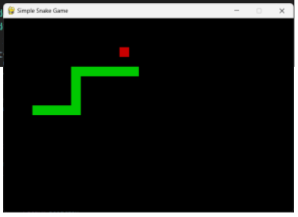
2022/Q4 (GPT3.5)

```
import pygame
import random

pygame.init()

# Constants
WIDTH, HEIGHT = 800, 600
GRIDSIZE = 20

# Snake class
class Snake:
    def __init__(self):
        self.length = 1
        self.positions = [((WIDTH/2), (HEIGHT/2))]
        self.direction = random.choice([UP, DOWN, LEFT, RIGHT
        self.color = GREEN

    def get_head_position(self):
        return self.positions[0]

    def move(self):
        cur = self.get_head_position()
        x, y = self.direction
        new = (((cur[0] + (x*GRIDSIZE)) % WIDTH),
                (cur[1] + (y*GRIDSIZE)) % HEIGHT)
```

```
$ python gpt3.5_snake.py
TypeError: 'float' object cannot
be interpreted as an integer
```
❌

2025/Q4 (GPT5)

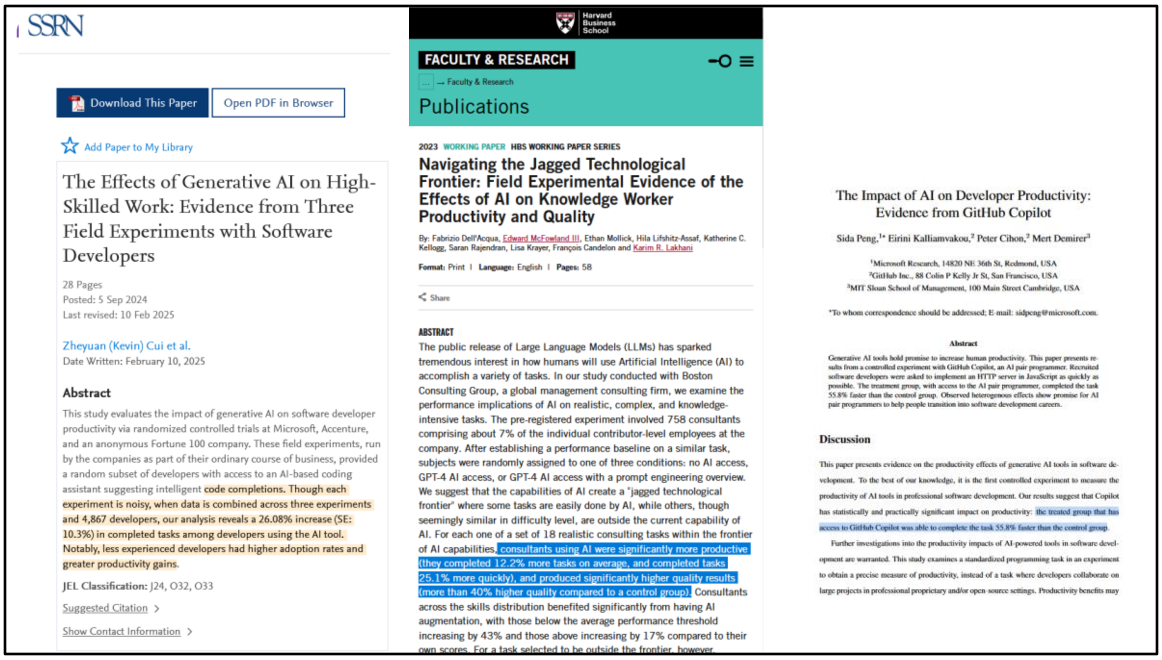
```
# Snake and food initialization
snake = [(100, 100), (80, 100), (60, 100)]
direction = (CELL_SIZE, 0)
food = (random.randrange(0, WIDTH, CELL_SIZE),
        random.randrange(0, HEIGHT, CELL_SIZE))

def draw_snake(snake):
    for segment in snake:
        pygame.draw.rect(screen, GREEN, (*segment, CELL_SIZE, CELL_SIZE))

def move_snake(snake, direction):
    head_x, head_y = snake[0]
    new_head = (head_x + direction[0], head_y + direction[1])
    return [new_head] + snake[:-1]

def check_collision(snake):
    head = snake[0]
    if (head[0] < 0 or head[0] >= WIDTH or
        head[1] < 0 or head[1] >= HEIGHT or head in snake[1:]):
        return True
    return False

def generate_food(snake):
    while True:
        position = (rand
                    rand
        if position not
            return posit
```



One-
shotted it
✅

Similar progress has happened in raw AI programming ability within similar timeframe.

GPT2 didn't even produce coherent text. GPT3.5 was already writing code, but it was often incorrect. Now, GPT5+ one-shots simple programming tasks almost without exception.


SSRN

Download This Paper | Open PDF in Browser

Add Paper to My Library

The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers

28 Pages
Posted: 5 Sep 2024
Last revised: 10 Feb 2025

Zheyuan (Kevin) Cui et al.
Date Written: February 10, 2025

Abstract

This study evaluates the impact of generative AI on software developer productivity via randomized controlled trials at Microsoft, Accenture, and an anonymous Fortune 100 company. These field experiments, run by the companies as part of their ordinary course of business, provided a random subset of developers with access to an AI-based coding assistant suggesting intelligent code completions. Though each experiment is noisy, when data is combined across three experiments and 4,867 developers, our analysis reveals a 26.08% increase (SE: 10.3%) in completed tasks among developers using the AI tool. Notably, less experienced developers had higher adoption rates and greater productivity gains.

JEL Classification: J24, O32, O33

Suggested Citation

Show Contact Information

Harvard Business School

FACULTY & RESEARCH

… → Faculty & Research

Publications

2023 WORKING PAPER HBS WORKING PAPER SERIES

Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality

By: Fabrizio Dell'Acqua, Edward McFowland III, Ethan Mollick, Hila Lifshitz-Assaf, Katherine C. Kellogg, Saran Rajendran, Lisa Krayer, François Candelon and Karim R. Lakhani

Format: Print | **Language:** English | **Pages:** 58

Share

ABSTRACT

The public release of Large Language Models (LLMs) has sparked tremendous interest in how humans will use Artificial Intelligence (AI) to accomplish a variety of tasks. In our study conducted with Boston Consulting Group, a global management consulting firm, we examine the performance implications of AI on realistic, complex, and knowledge-intensive tasks. The pre-registered experiment involved 758 consultants comprising about 7% of the individual contributor-level employees at the company. After establishing a performance baseline on a similar task, subjects were randomly assigned to one of three conditions: no AI access, GPT-4 AI access, or GPT-4 AI access with a prompt engineering overview. We suggest that the capabilities of AI create a "jagged technological frontier" where some tasks are easily done by AI, while others, though seemingly similar in difficulty level, are outside the current capability of AI. For each one of a set of 18 realistic consulting tasks within the frontier of AI capabilities, consultants using AI were significantly more productive (they completed 12.2% more tasks on average, and completed tasks 25.1% more quickly), and produced significantly higher quality results (more than 40% higher quality compared to a control group). Consultants across the skills distribution benefited significantly from having AI augmentation, with those below the average performance threshold increasing by 43% and those above increasing by 17% compared to their own scores. For a task selected to be outside the frontier, however

The Impact of AI on Developer Productivity: Evidence from GitHub Copilot

Sida Peng,[1*] Eirini Kalliamvakou,[2] Peter Cihon,[2] Mert Demirer[3]

[1]Microsoft Research, 14820 NE 36th St, Redmond, USA
[2]GitHub Inc., 88 Colin P Kelly Jr St, San Francisco, USA
[3]MIT Sloan School of Management, 100 Main Street Cambridge, USA

[*]To whom correspondence should be addressed; E-mail: sidpeng@microsoft.com.

Abstract

Generative AI tools hold promise to increase human productivity. This paper presents results from a controlled experiment with GitHub Copilot, an AI pair programmer. Recruited software developers were asked to implement an HTTP server in JavaScript as quickly as possible. The treatment group, with access to the AI pair programmer, completed the task 55.8% faster than the control group. Observed heterogenous effects show promise for AI pair programmers to help people transition into software development careers.

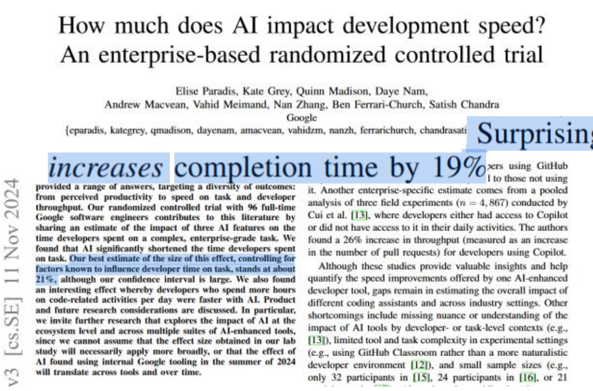
Discussion

This paper presents evidence on the productivity effects of generative AI tools in software development. To the best of our knowledge, it is the first controlled experiment to measure the productivity of AI tools in professional software development. Our results suggest that Copilot has statistically and practically significant impact on productivity: the treated group that has access to GitHub Copilot was able to complete the task 55.8% faster than the control group.

Further investigations into the productivity impacts of AI-powered tools in software development are warranted. This study examines a standardized programming task in an experiment to obtain a precise measure of productivity, instead of a task where developers collaborate on large projects in professional proprietary and/or open-source settings. Productivity benefits may

Research on the effects to work productivity are starting to emerge: General consultants were 25% faster and at the same time quality improved by 40%! Coders completed simple programming tasks in half the time.

Tampereen yliopisto



How much does AI impact development speed?
An enterprise-based randomized controlled trial

Elise Paradis, Kate Grey, Quinn Madison, Daye Nam,
Andrew Macvean, Vahid Meimand, Nan Zhang, Ben Ferrari-Church, Satish Chandra
Google
{eparadis, kategrey, qmadison, dayenam, amacvean, vahidzm, nanzh, ferrarichurch, chandrasati

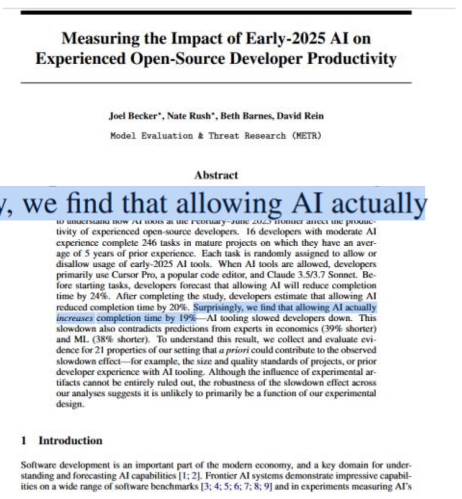
v3 [cs.SE] 11 Nov 2024

provided a range of answers, targeting a diversity of outcomes: from perceived productivity to speed on task and developer throughput. Our randomized controlled trial with 96 full-time Google software engineers contributes to this literature by sharing an estimate of the impact of three AI features on the time developers spent on a complex, enterprise-grade task. We found that AI significantly shortened the time developers spent on task. Our best estimate of the size of this effect, controlling for factors known to influence developer time on task, stands at about 21%, although our confidence interval is large. We also found an interesting effect whereby developers who spend more hours on code-related activities per day were faster with AI. Product and future research considerations are discussed. In particular, we invite further research that explores the impact of AI at the ecosystem level and across multiple suites of AI-enhanced tools, since we cannot assume that the effect size obtained in our lab study will necessarily apply more broadly, or that the effect of AI found using internal Google tooling in the summer of 2024 will translate across tools and over time.

pers using GitHub l to those not using it. Another enterprise-specific estimate comes from a pooled analysis of three field experiments ($n = 4,867$) conducted by Cui et al. [13], where developers either had access to Copilot or did not have access to it in their daily activities. The authors found a 26% increase in throughput (measured as an increase in the number of pull requests) for developers using Copilot.

Although these studies provide valuable insights and help quantify the speed improvements offered by one AI-enhanced developer tool, gaps remain in estimating the overall impact of different coding assistants and across industry settings. Other shortcomings include missing nuance or understanding of the impact of AI tools by developer- or task-level contexts (e.g., [13]), limited tool and task complexity in experimental settings (e.g., using GitHub Classroom rather than a more naturalistic developer environment [12]), and small sample sizes (e.g., only 32 participants in [15], 24 participants in [16], or 21

Surprisingly, we find that allowing AI actually *increases* completion time by 19%



Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity

Joel Becker*, Nate Rush*, Beth Barnes, David Rein

Model Evaluation & Threat Research (METR)

Abstract

to understand how AI tools at the February–June 2025 frontier affect the productivity of experienced open-source developers. 16 developers with moderate AI experience complete 246 tasks in mature projects on which they have an average of 5 years of prior experience. Each task is randomly assigned to allow or disallow usage of early-2025 AI tools. When AI tools are allowed, developers primarily use Cursor Pro, a popular code editor, and Claude 3.5/3.7 Sonnet. Before starting tasks, developers forecast that allowing AI will reduce completion time by 24%. After completing the study, developers estimate that allowing AI reduced completion time by 20%. Surprisingly, we find that allowing AI actually *increases* completion time by 19%—AI tooling slowed developers down. This slowdown also contradicts predictions from experts in economics (39% shorter) and ML (38% shorter). To understand this result, we collect and evaluate evidence for 21 properties of our setting that *a priori* could contribute to the observed slowdown effect—for example, the size and quality standards of projects, or prior developer experience with AI tooling. Although the influence of experimental artifacts cannot be entirely ruled out, the robustness of the slowdown effect across our analyses suggests it is unlikely to primarily be a function of our experimental design.

1 Introduction

Software development is an important part of the modern economy, and a key domain for understanding and forecasting AI capabilities [1; 2]. Frontier AI systems demonstrate impressive capabilities on a wide range of software benchmarks [3; 4; 5; 6; 7; 8; 9] and in experiments measuring AI's

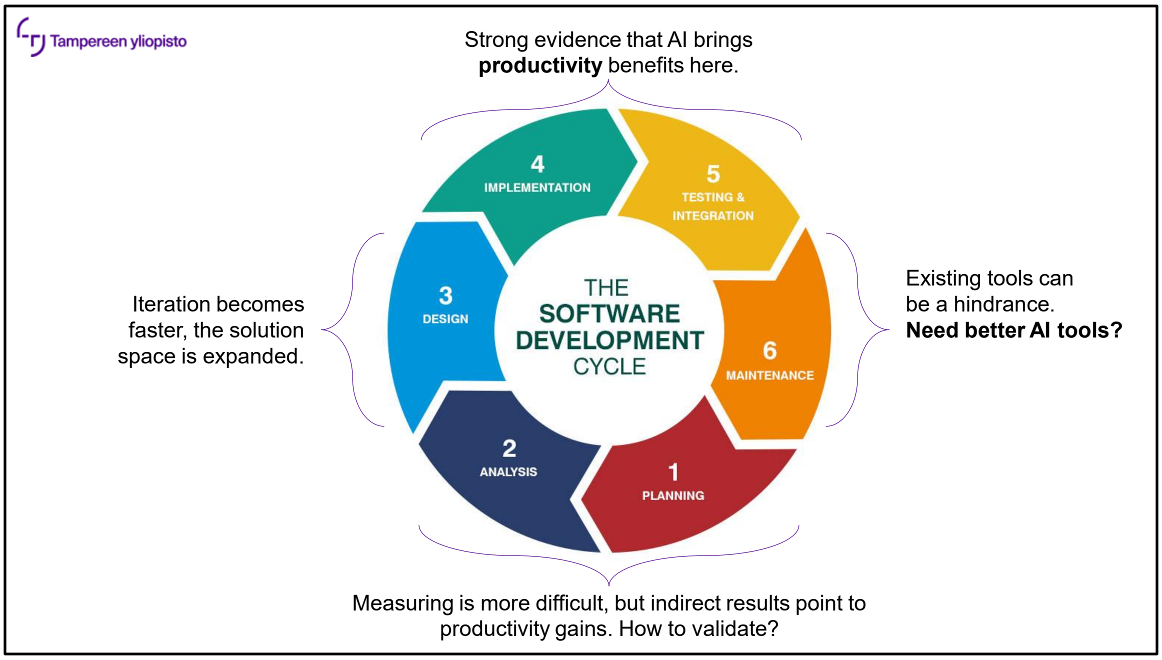
+ "… suggesting that participants in our sample with increased levels of seniority were faster with AI than those with lower levels of seniority."

https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study

It would seem that green-field development tasks are completed 10-50% faster with the help of artificial intelligence, depending on the context and complexity of the tasks.

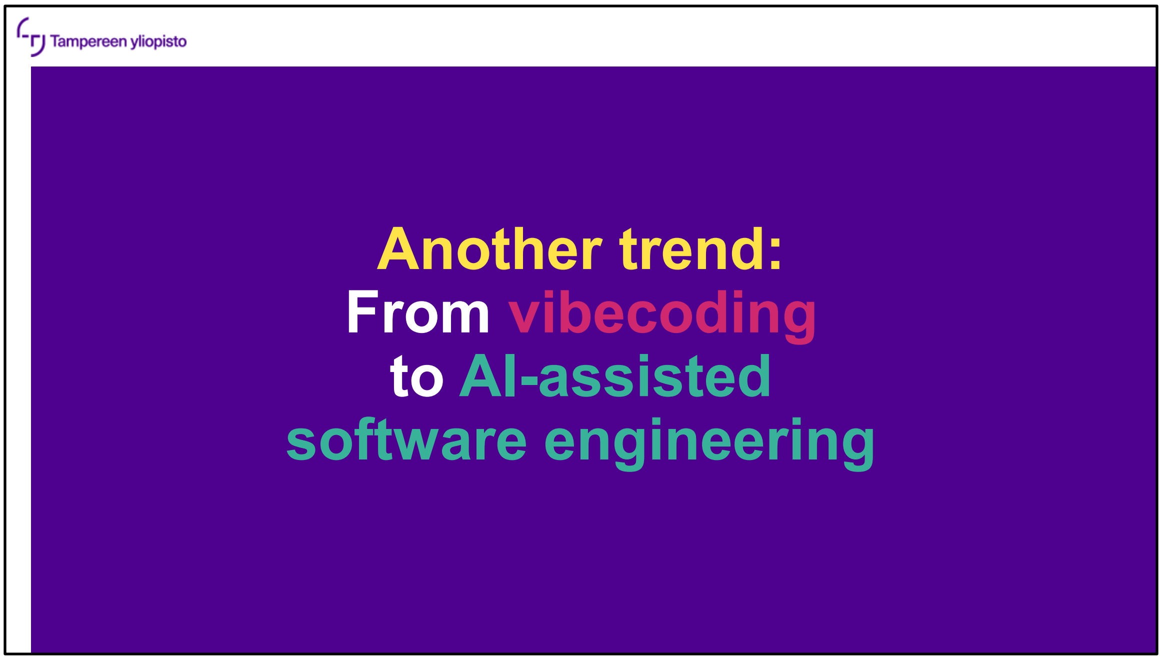
There is also few opposite results: In the study in question, there were 16 developers who did about 250 tasks (bug fixes, refactoring, additional features) in large and complex projects (1M+ lines of code) that they knew well (5+ years of experience with the project).

Possible causes:
"they also spend a somewhat higher proportion of their time idle"
"only 44% of the participants had ever used Cursor (AI coding tool) before the study"


Tampereen yliopisto
Strong evidence that AI brings **productivity** benefits here.
4 IMPLEMENTATION
5 TESTING & INTEGRATION
3 DESIGN
THE SOFTWARE DEVELOPMENT CYCLE
6 MAINTENANCE
2 ANALYSIS
1 PLANNING
Iteration becomes faster, the solution space is expanded.
Existing tools can be a hindrance. **Need better AI tools?**
Measuring is more difficult, but indirect results point to productivity gains. How to validate?

As mentioned before, there is already evidence that AI is useful especially in design, implementation and testing. And to some extent planning and analysis. For maintenance, the SDLC the benefits are inconclusive, mostly because of lack of tooling.

Still, it is good to acknowledge that AI tooling for SWE is not just generating code. It is automating work in every phase of SDLC!

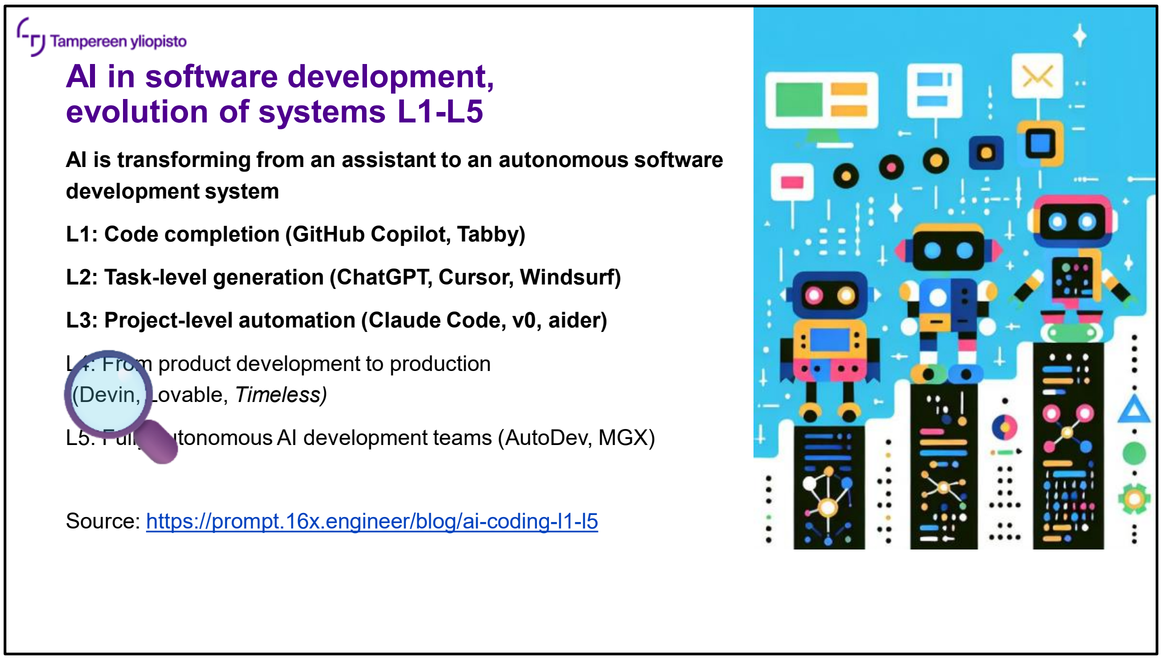


Tampereen yliopisto

Another trend: From vibecoding to AI-assisted software engineering

One could even claim that automated generation of code has already been solved in cases where the situation can be explained to artificial intelligence. Such “vibe coding”, i.e., programming by using natural language, has proven useful, especially if you know what you want and can describe the requirements of the system for the AI in sufficient detail.

Now, our scope is growing away from only coding: we are starting to solve problems in software production and engineering (maintainability, complexity, information sharing, architecture, etc., etc.).



Tampereen yliopisto

AI in software development, evolution of systems L1-L5

AI is transforming from an assistant to an autonomous software development system

L1: Code completion (GitHub Copilot, Tabby)

L2: Task-level generation (ChatGPT, Cursor, Windsurf)

L3: Project-level automation (Claude Code, v0, aider)

L4: From product development to production (Devin, Lovable, *Timeless)*

L5: Fully autonomous AI development teams (AutoDev, MGX)

Source: https://prompt.16x.engineer/blog/ai-coding-l1-l5

The idea of L1–L5 levels is borrowed from self-driving cars, and it is useful also for AI SWE. At first, AI only assists, but the higher the level, the bigger portion of SWE process it automates and eventually works almost autonomously.

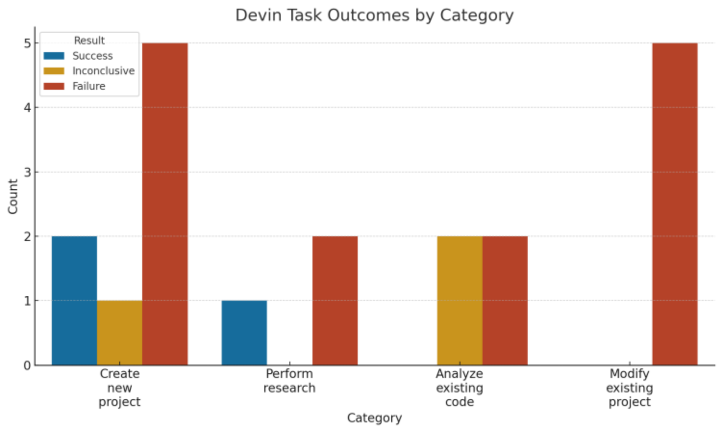
Where are we now?
L1 and L2 are already in use – many software companies are already using GitHub Copilot (or similar) or ChatGPT to support coding. L3, or project-level automation, is developing rapidly, but still requires a lot of human guidance. L4 and L5, on the other hand, hope to provide multiagent systems of AI teams, but they are not yet fully ready for widespread use. L5 systems even find their own problems to solve within the organizations they are deployed in.

What does this mean for developers and businesses? L1-L2 speed up routines, L3 brings efficiency to move entire projects forward, and L4-L5 can revolutionize who develops software and how and where software is created.

As a research group, our goal is to study and find out how software development is changing. We are currently building our own L4 system called Timeless.

Tampereen yliopisto

Promises vs. Reality (case *Devin*)

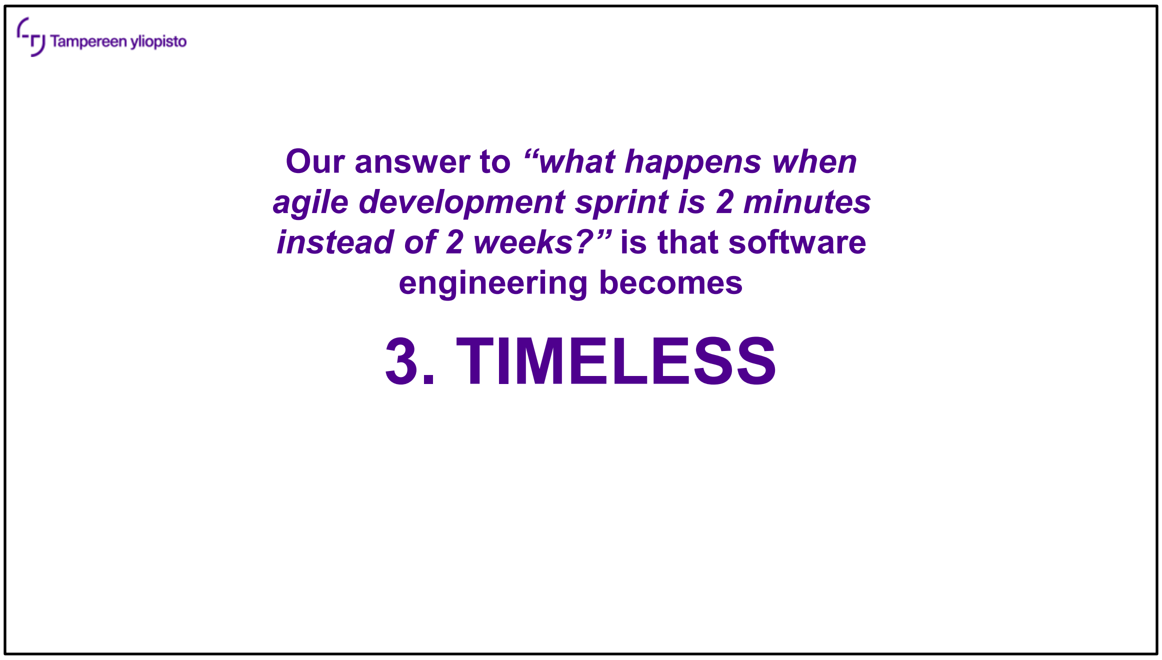


Source https://www.answer.ai/posts/2025-01-08-devin.html
Answer.AI evaluated Devini on 20 different tasks on January 2025.

However, currently, the "ugly" truth is that many tools cannot deliver on their promises: For example, the L4 called Devin rarely worked in early 2025. Of the 20 tasks Answer.AI gave for it to solve, 14 failed, 3 with unclear results, and only 3 successes.

More problematic was the difficulty in predicting which tasks it could tackle: based on the experiment, the tasks would fail in complex and time-consuming ways. The seemingly promising autonomous nature became a burden - Devin spent long periods of time looking for impossible solutions instead of identifying fundamental obstacles.

One has to evaluate every AI tool and make sure they are a good fit for the kind of problems one is trying to solve.

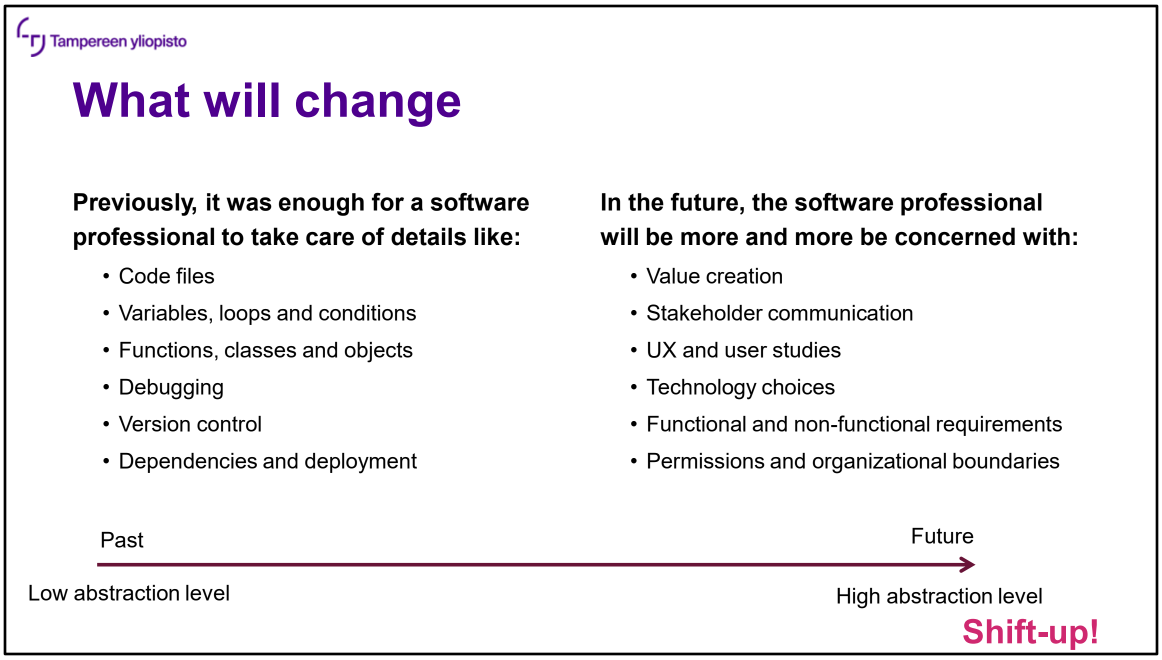


Tampereen yliopisto

Our answer to *“what happens when agile development sprint is 2 minutes instead of 2 weeks?”* is that software engineering becomes

3. TIMELESS

GPT-Lab has inspirations to innovate in this space. Our answer to “what happens when agile development sprint is 2 minutes instead of 2 weeks?” is timeless.


Tampereen yliopisto

What will change

Previously, it was enough for a software professional to take care of details like:

- Code files
- Variables, loops and conditions
- Functions, classes and objects
- Debugging
- Version control
- Dependencies and deployment

In the future, the software professional will be more and more be concerned with:

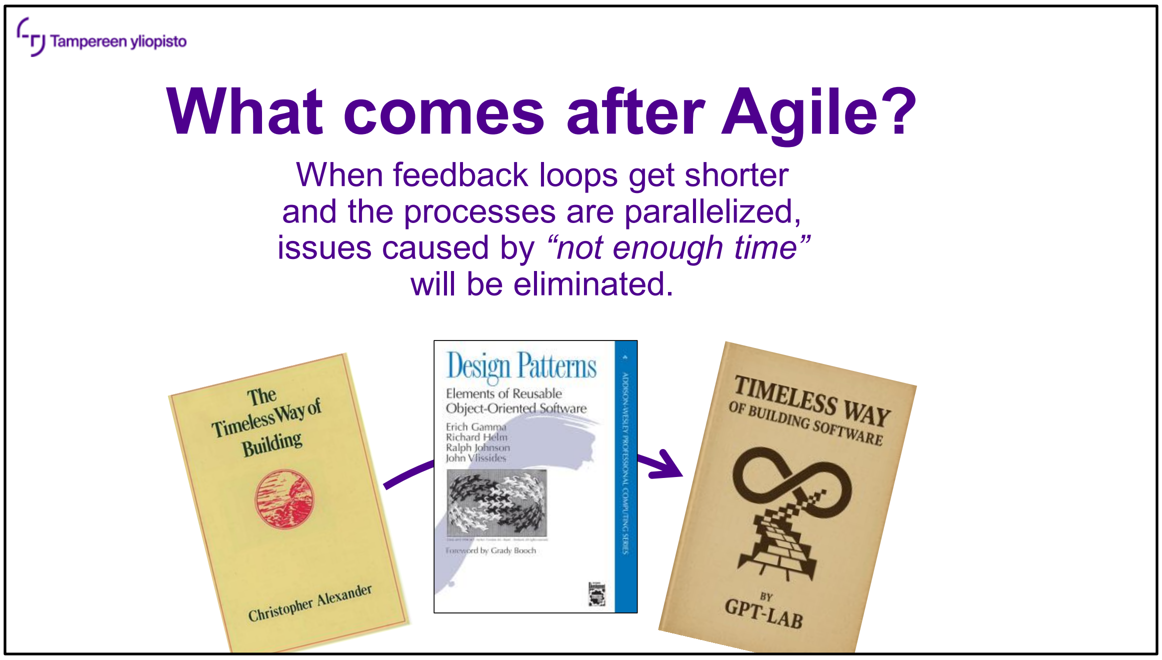
- Value creation
- Stakeholder communication
- UX and user studies
- Technology choices
- Functional and non-functional requirements
- Permissions and organizational boundaries

So that we can use the understanding of how software is made accumulated over decades. We still need experts who know how to make software. The level of abstraction only increases.

The main focus will be on the interaction and developer experience (AI-to-human interaction).

“The same pattern will play out with software engineers. We’ll see more people doing software engineering work and in a decade or so, what “software engineer” means will have transformed. Consider the restaurant owner from earlier who uses AI to create custom inventory software that is useful only for them. They won’t call themselves a software engineer.”

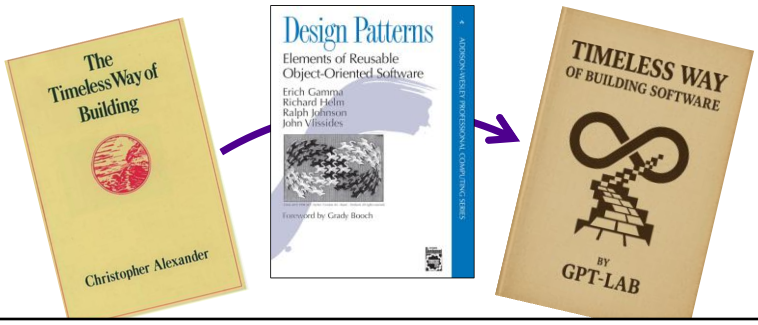
Vlad will discuss this in more formally in his Shift-up presentation.



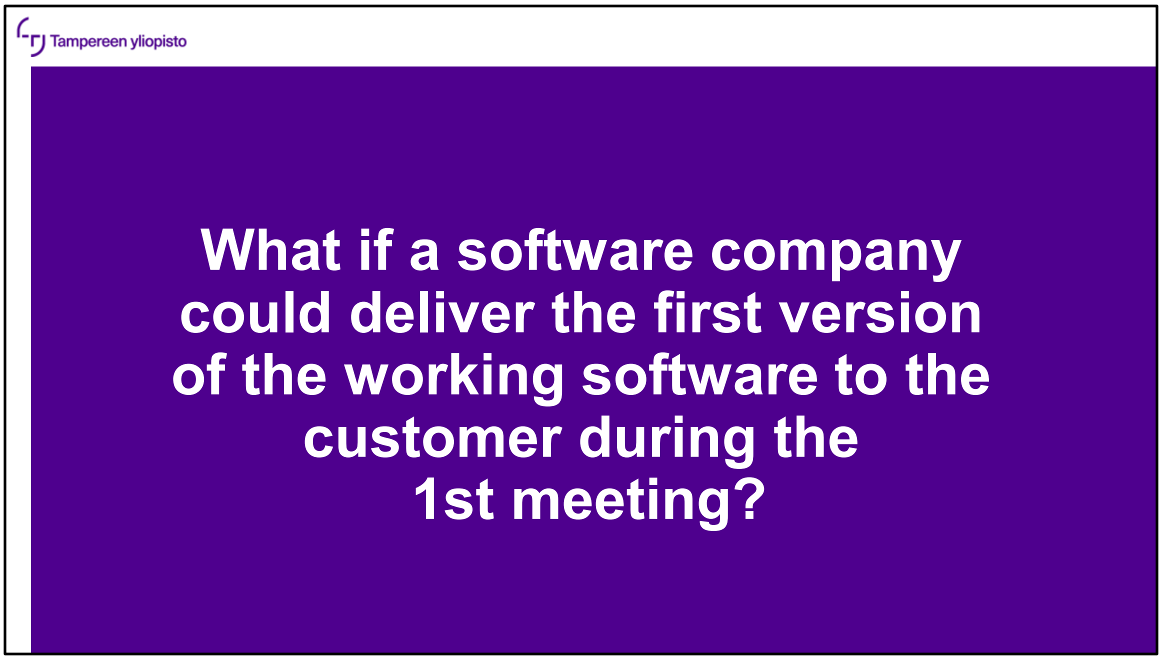
Tampereen yliopisto

What comes after Agile?

When feedback loops get shorter
and the processes are parallelized,
issues caused by *"not enough time"*
will be eliminated.



We believe that the focus shifts to prioritization, doing the right things, communication. An end product that works continuously and can be tested would still be essential. Quality improves due to the scope of testing, and challenges arise from versioning and integrations, deployment resistance to change, and training issues.



Tampereen yliopisto

What if a software company could deliver the first version of the working software to the customer during the 1st meeting?

With Timeless system, which is a realization of the Timeless idea, we ask....

Tampereen yliopisto

Ideas are plenty, so we also need to construct a Timeless system to show the idea works!

arXiv:2411.08507v1 [cs.SE] 13 Nov 2024

TimeLess: A Vision for the Next Generation of Software Development

Zeeshan Rasheed, Malik Abdul Sami, Jussi Rasku, Kai-Kristian Kemell, Zheying Zhang, Janne Harjamäki, Shahbaz Siddeeq, Sami Lahti, Tomas Herda, Mikko Nurminen, Niklas Lavesson, José Siqueira de Cerqueira, Toufique Hasan, Ayman Khan, Mahade Hasan, Mika Saari, Petri Rantanen, Jari Soini and Pekka Abrahamsson

Faculty of Information Technology and Communication Science, Tampere University

Tampere, Finland

ABSTRACT

[illegible]

KEYWORDS

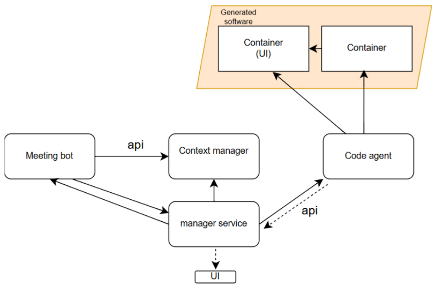
Artificial Intelligence, Natural Language Processing, Generative AI, Large Language Model, Software Engineering

[illegible]

Figure 1: Future Generation of Software Development

1 INTRODUCTION

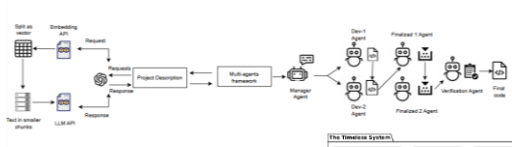
[illegible]

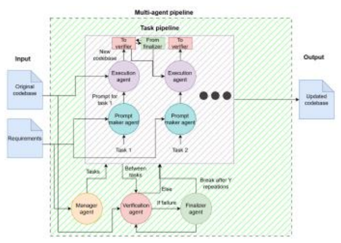


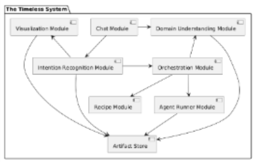
https://arxiv.org/abs/2411.08507

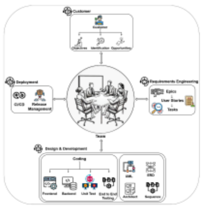
Tampereen yliopisto

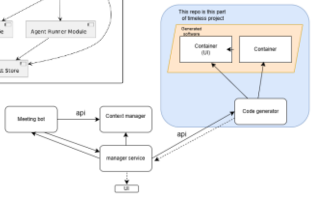
And this requires research











Chauhan, S. et al. *LLM-Generated Microservice Implementations from RESTful API Definitions*. Accepted to ENASE.

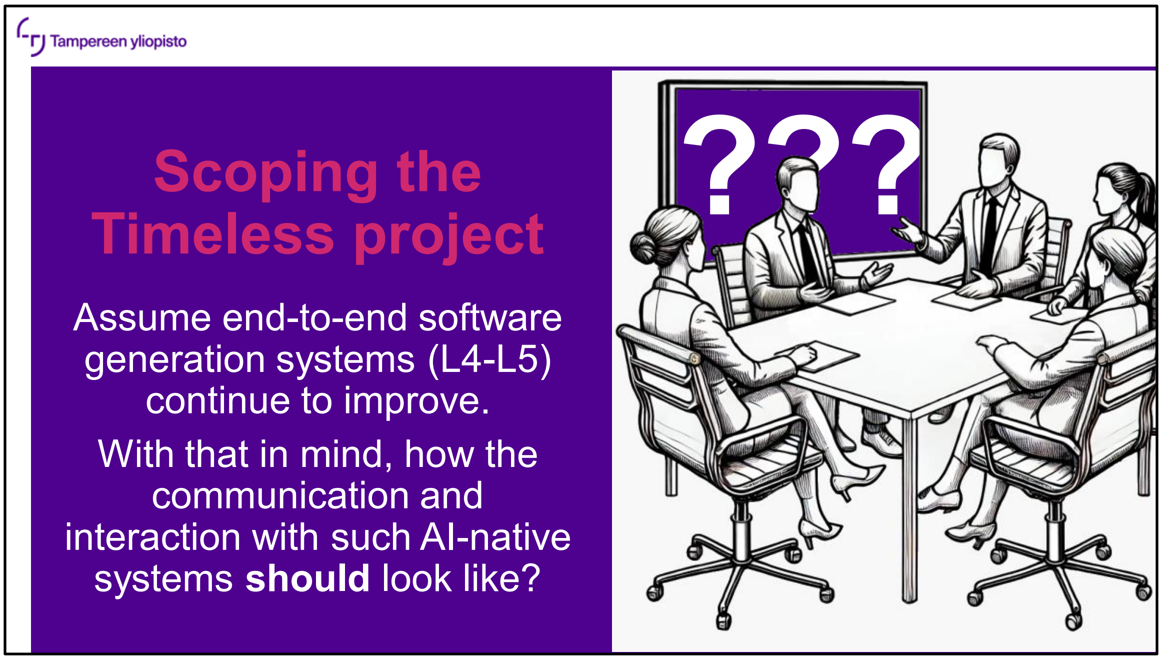
Ala-Salmi, V. et al. *Autonomous Legacy Web Application Upgrades Using a Multi-Agent System*. Accepted to ENASE 2025.

Khan, AA. et al. *Developing Retrieval Augmented Generation (RAG) based LLM Systems from PDFs: An Experience Report*. arXiv preprint.

Rasheed, Z. et al. *Large Language Models for Code Generation: The Practitioners Perspective*. Submitted to XP 2025.

Rasheed, Z. et al. *TimeLess: A Vision for the Next Generation of Software Development*. Submitted to XP 2025.

…..



Tampereen yliopisto

Scoping the Timeless project

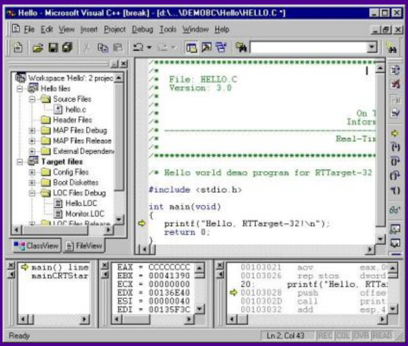
Assume end-to-end software generation systems (L4-L5) continue to improve.

With that in mind, how the communication and interaction with such AI-native systems **should** look like?

That is, with Timeless we concentrate on the post-Agile developer experience (devEX).

Tampereen yliopisto

It is time to let the go of the IDE paradigm



```
/**************************************
/*
/*  File: HELLO.C
/*  Version: 3.0
/*
/*
/*                                  On T
/*                                Inform
/* -------------------------------------
/*                              Real-Tim
/*
/**************************************

/* Hello world demo program for RTTarget-32

#include <stdio.h>

int main(void)
{
   printf("Hello, RTTarget-32!\n");
   return 0;
}
```

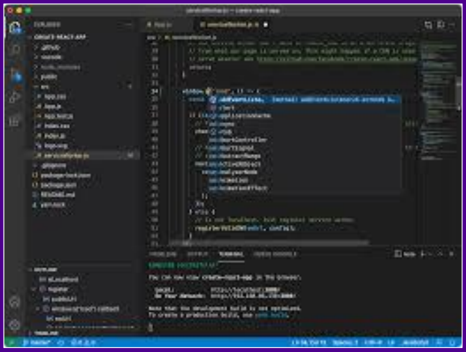
```
main() line
mainCRTStar
```

```
EAX = CCCCCCCC
EBX = 00041390
ECX = 00000000
EDX = 00136E40
ESI = 00000040
EDI = 00135F3C
```

```
00103021   mov        eax,0
00103026   rep stos   dword
20:        printf("Hello, RTTa
00103028   push       offse
0010302D   call       print
00103032   add        esp,4
```

1998

25+ years of “progress”



2025

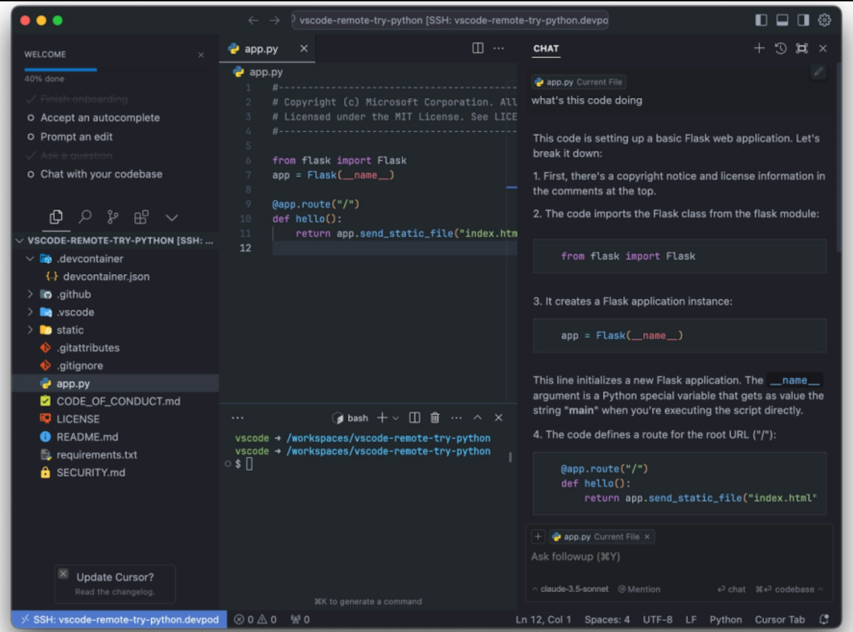
Surprisingly little has changed in the tools we have used to create software. It seems that in the quarter of a century of innovation of the IDE we have only managed to go from 4/3 to 16/9 aspect ratio.

Of course, there has been development, but that has been evolution, not revolution. I feel like this is changing!

Tampereen yliopisto

Surely not like this?

(that is, smarter autocomplete and ability to chat on code and ask for one-shot edits)



```python
#-------------------------------------------------
# Copyright (c) Microsoft Corporation. All
# Licensed under the MIT License. See LICE
#-------------------------------------------------

from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return app.send_static_file("index.htm
```

```
vscode → /workspaces/vscode-remote-try-python
vscode → /workspaces/vscode-remote-try-python
$ 
```

what's this code doing

This code is setting up a basic Flask web application. Let's break it down:

1. First, there's a copyright notice and license information in the comments at the top.

2. The code imports the Flask class from the flask module:

```python
from flask import Flask
```

3. It creates a Flask application instance:

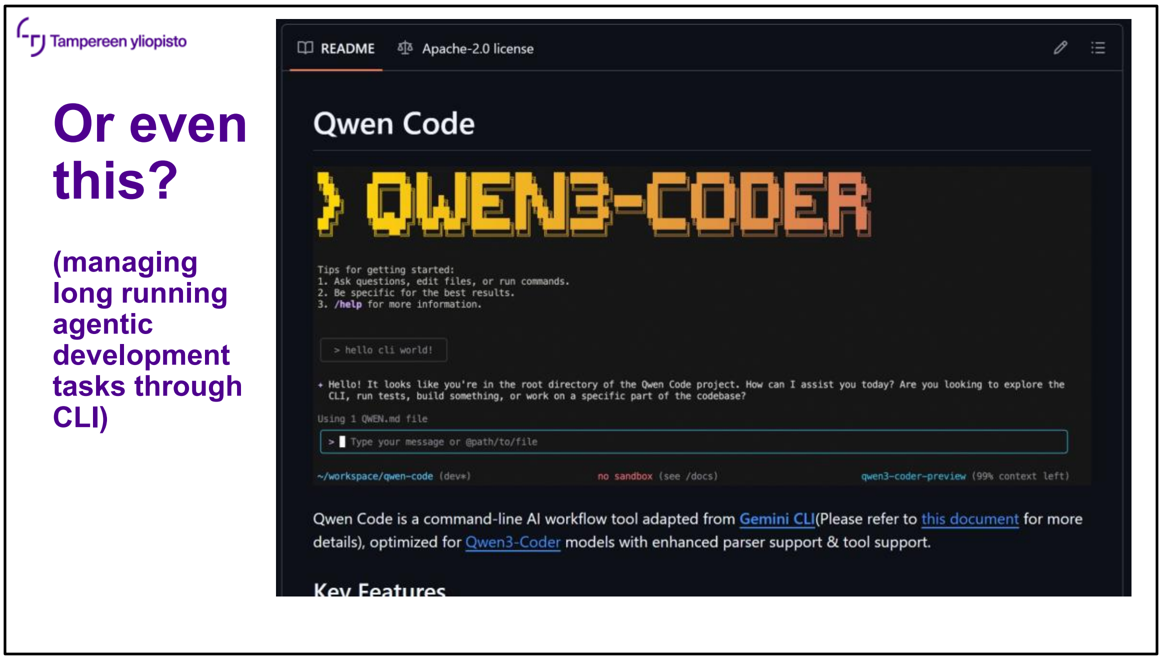
```python
app = Flask(__name__)
```

This line initializes a new Flask application. The `__name__` argument is a Python special variable that gets as value the string "**main**" when you're executing the script directly.

4. The code defines a route for the root URL ("/"):

```python
@app.route("/")
def hello():
    return app.send_static_file("index.html"
```

The L1-L2 systems that assist in producing small snippets of code work well within the IDE paradigm, but I feel the AI technology can move further.



Tampereen yliopisto

Or even this?

(managing long running agentic development tasks through CLI)

README | Apache-2.0 license

Qwen Code

```
> QWEN3-CODER

Tips for getting started:
1. Ask questions, edit files, or run commands.
2. Be specific for the best results.
3. /help for more information.

 > hello cli world!

+ Hello! It looks like you're in the root directory of the Qwen Code project. How can I assist you today? Are you looking to explore the
  CLI, run tests, build something, or work on a specific part of the codebase?

Using 1 QWEN.md file
 > Type your message or @path/to/file

~/workspace/qwen-code (dev*)              no sandbox (see /docs)              qwen3-coder-preview (99% context left)
```

Qwen Code is a command-line AI workflow tool adapted from Gemini CLI(Please refer to this document for more details), optimized for Qwen3-Coder models with enhanced parser support & tool support.

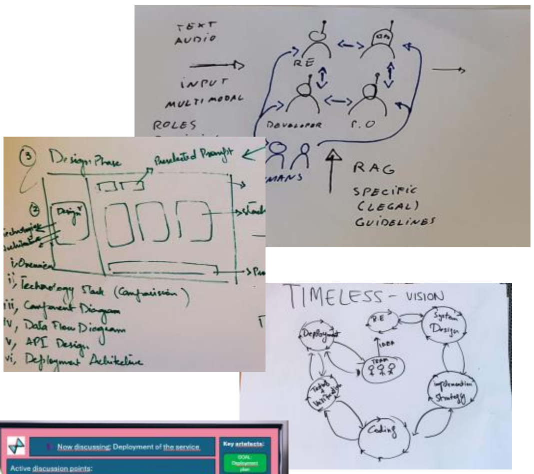
Key Features

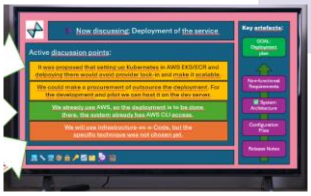
Agentic programming tools such as Claude code, Codex CLI and others allow you to command multiple AI agents operating directly on the codebase without the IDE on your way. You still interact with code from time to time but mostly work as a supervisor and reviewer rather than developer.

Tampereen yliopisto

Our thoughts

- **Future is in Human-AI voice interaction**
- **AI systems still follow SDLC stages and best practices**
- **System needs to record any decisions** (traceability)
- **It is an *iterative* end-to-end system**: It generates code, runs it, and allows iteration




Now discussing: Deployment of the service

Active discussion points:

It was proposed that setting up Kubernetes in AWS EKS/ECR and deploying there would avoid provider lock-in and make it scalable.

We could make a procurement of outsource the deployment. For the development and pilot we can host it on the dev server.

We already use AWS, so the deployment is to be done there, the system already has AWS CLI access.

We will use Infrastructure-as-a-Code, but the specific technique was not chosen yet.

Key artefacts:

GOAL: Deployment plan

Non-functional Requirements

✓ System Architecture

Configuration Files

Release Notes

Our Timeless idea is to have meetings **turn into runnable software** by the end of the session. Instead of meetings that produce plans, it uses AI agents that listen, transcribe, summarize, extract user stories/requirements, generate design, code, tests, and even deployable artifacts in (near) real time based on discussion. Human participants steer decisions while the system executes engineering tasks as they speak, shifting SE toward real-time action rather than post-meeting work. We have a prototype system we are hoping to allow others to try soon.

For now, we need to rely on video (I’m not brave enough to have a live demo here, but can give it after the talk later to you).

Tampereen yliopisto
Company data
Company data
Company dat
adapter (MCP
servers
Mee
Intention and
consensus detectio
(MCP) service
Oliopäivät : Propose your own Ti-
meless microservice or subsystem
Artefact
store
Designer
(MCP) service
e
n

https://github.com/GPT-Laboratory/timeless-archit

There are many ways one can collaborate in realizing Timeless. We are open for contributions on MCP servers or microservices that can extend the core Timeless system.

Tampereen yliopisto

@JussiRasku

jussi . rasku
@ tuni . fi

@yorak

0000-0002-
4401-8013

jussi-rasku-
91a5074

Key takeaways:

1. AI is already **several orders of magnitude faster** at coding than us.
2. AI is also **catching up with human capabilities** in pure software engineering skill.
3. Hence, soon, the **software is developed at a higher level of abstraction.**
4. Agile is followed by **Timeless Software Engineering**, or something similar.