Software Engineering Automation: From early tools to Generative AI and beyond

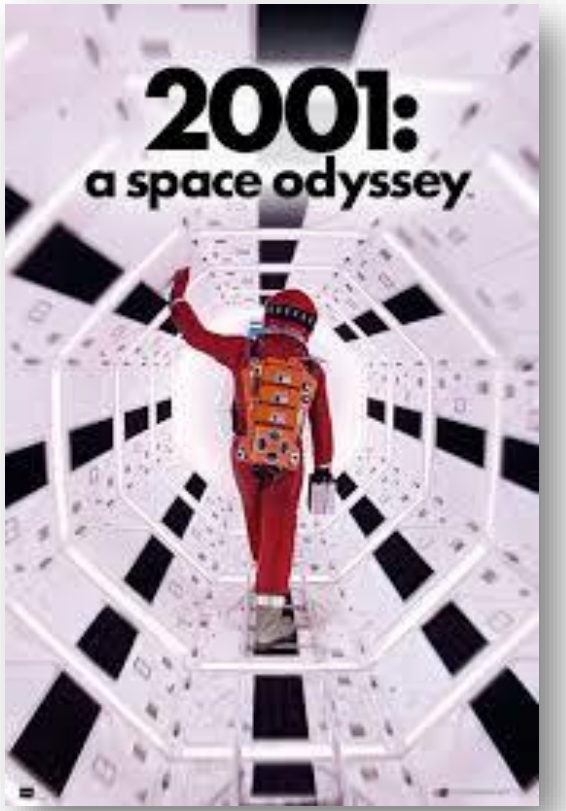
SFSCon 2024

November 8th, 2024

Who am I?

Jorge Melegati

Human development is about tools

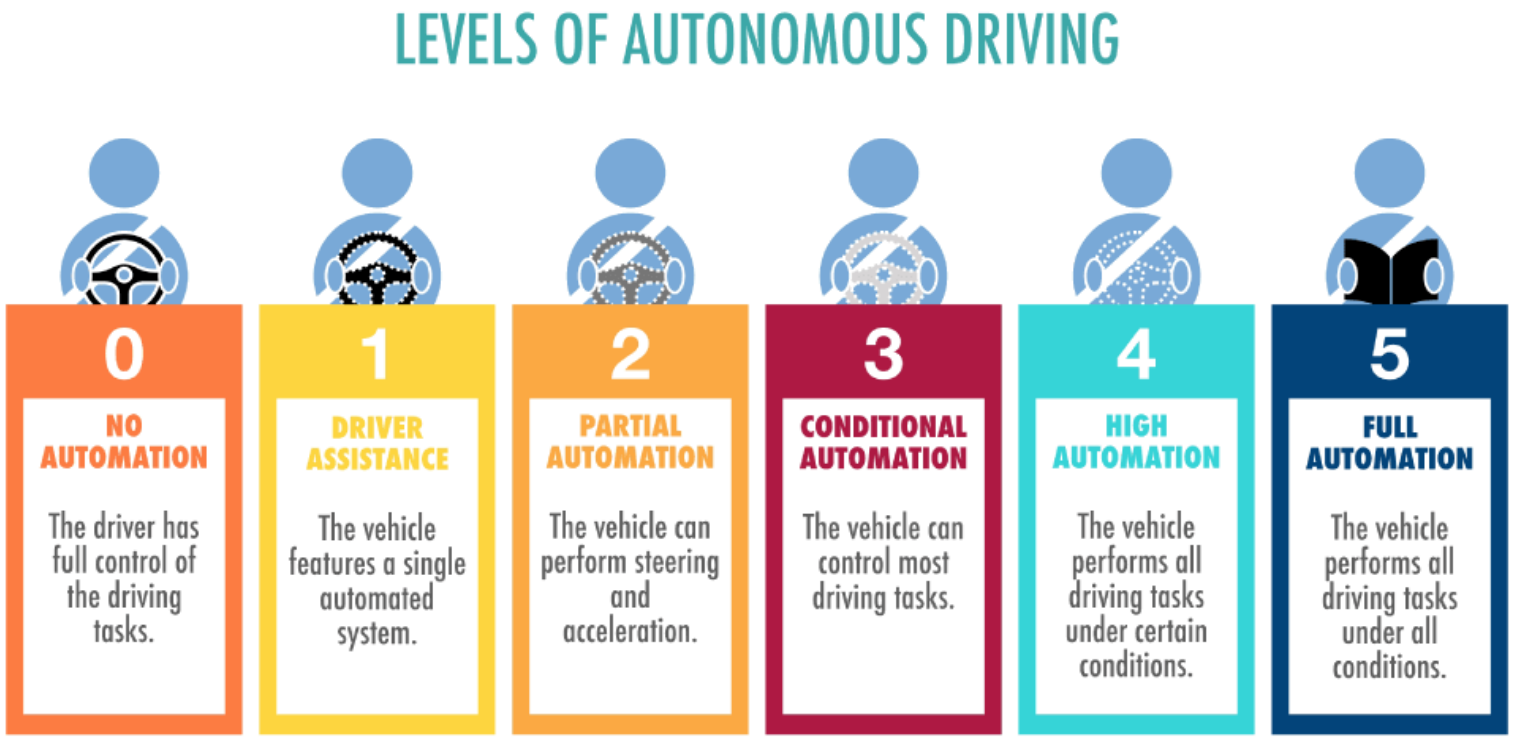


More evident with automation

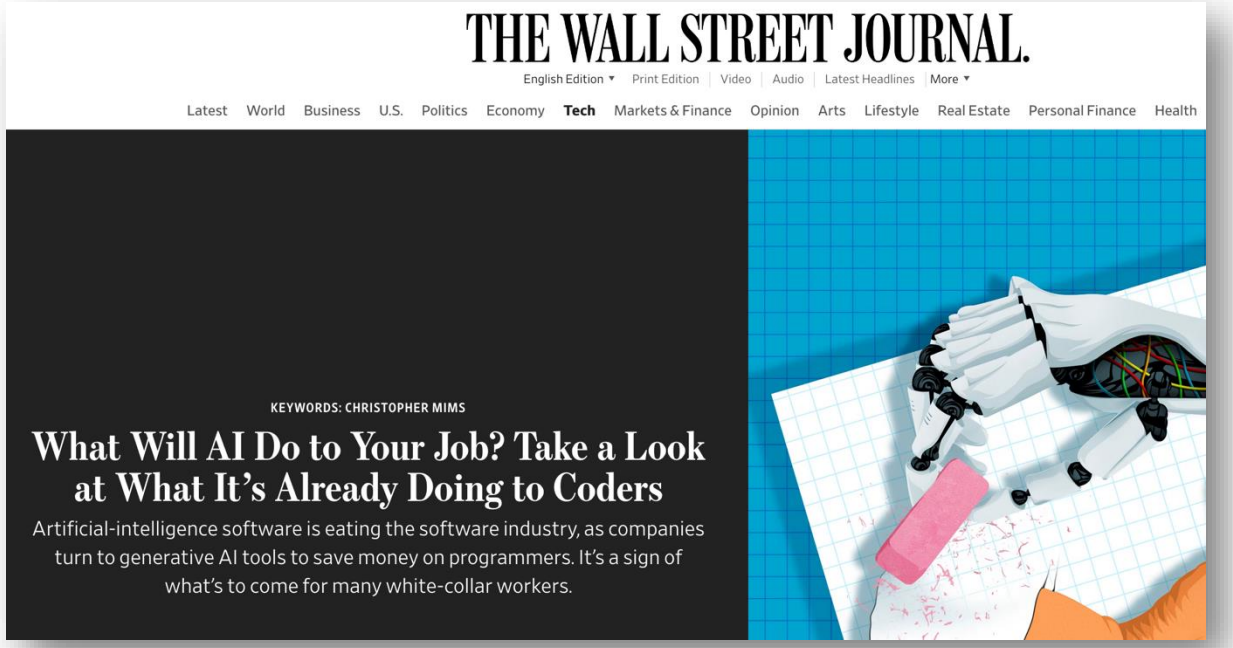
Before the invention of the press by Gutenberg, a scribe took several months to years to copy a bible!

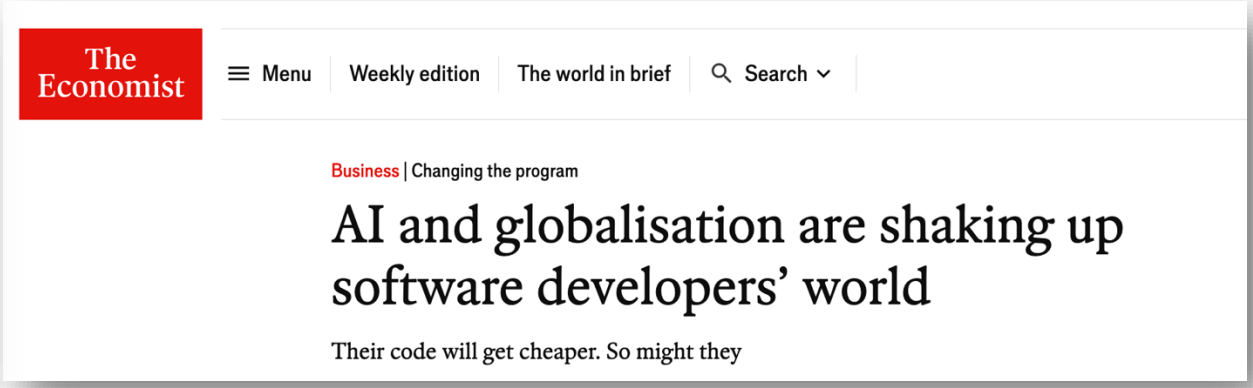
But automation is an evolving process…

Example: Driving automation levels proposed by SAE



Generative AI highlighted the possibility of automating software engineering



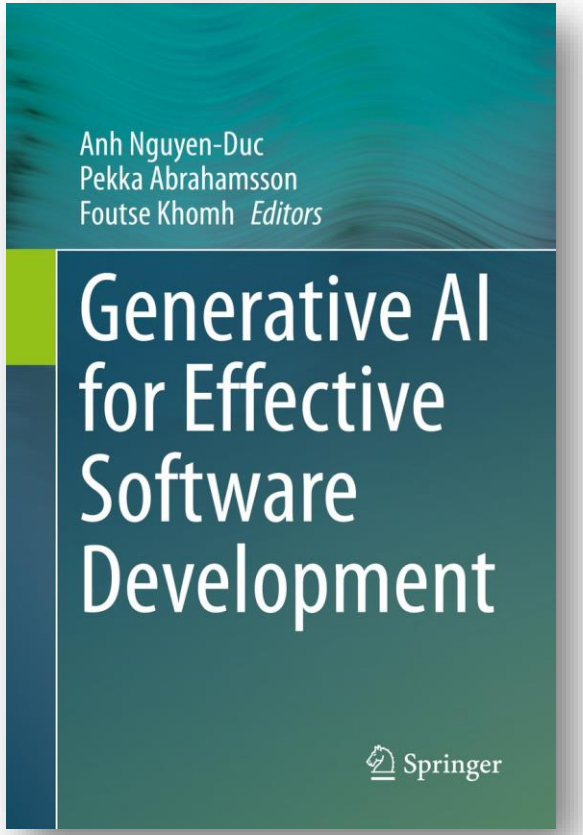


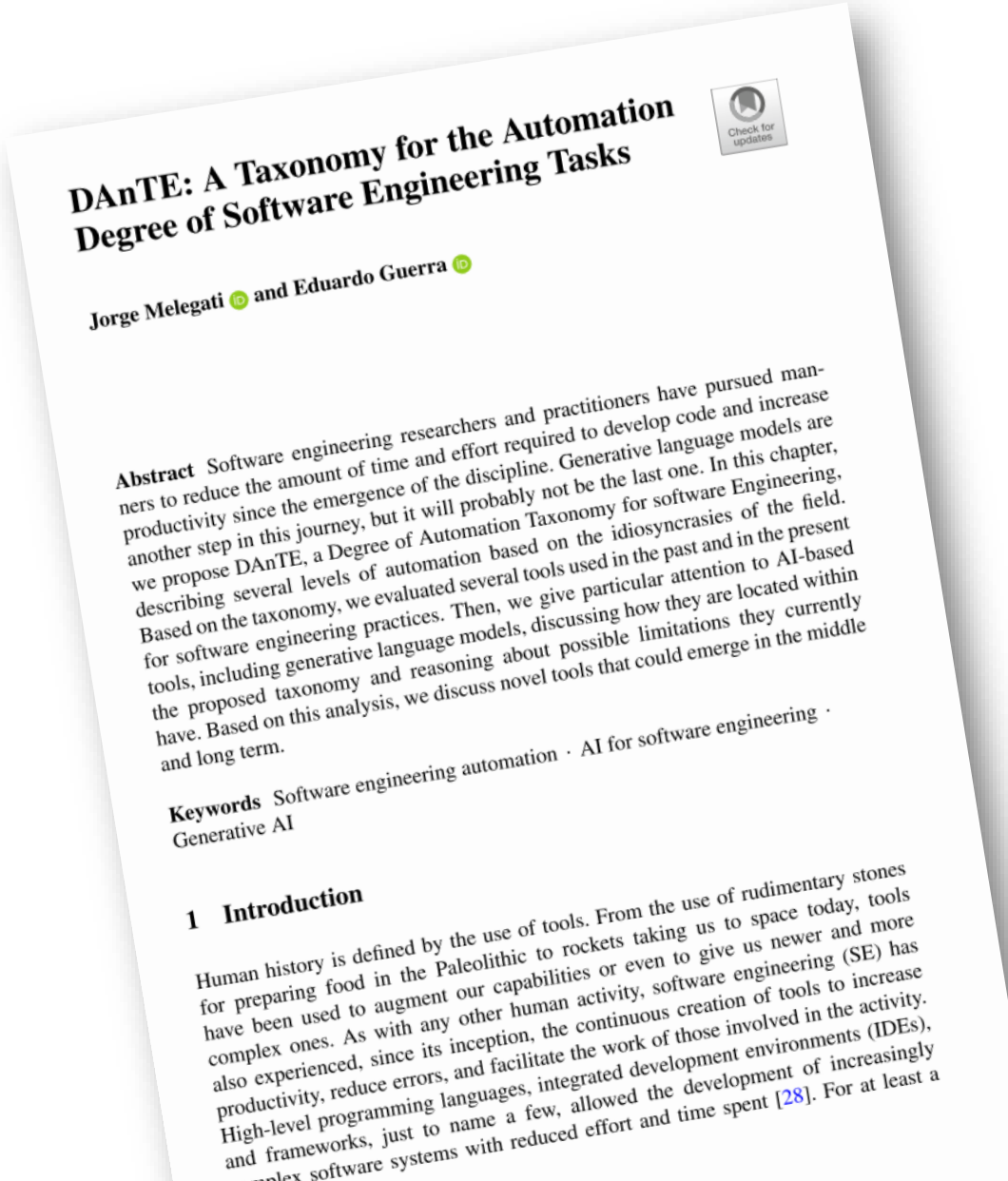
But is automating software engineering a new thing?

But is automating software engineering a new thing?

NO!

Historically, we can see that automation of software engineering has followed some steps!





DAnTE: A Taxonomy for the Automation Degree of Software Engineering Tasks

Jorge Melegati and Eduardo Guerra

Abstract Software engineering researchers and practitioners have pursued manners to reduce the amount of time and effort required to develop code and increase productivity since the emergence of the discipline. Generative language models are another step in this journey, but it will probably not be the last one. In this chapter, we propose DAnTE, a Degree of Automation Taxonomy for software Engineering, describing several levels of automation based on the idiosyncrasies of the field. Based on the taxonomy, we evaluated several tools used in the past and in the present for software engineering practices. Then, we give particular attention to AI-based tools, including generative language models, discussing how they are located within the proposed taxonomy and reasoning about possible limitations they currently have. Based on this analysis, we discuss novel tools that could emerge in the middle and long term.

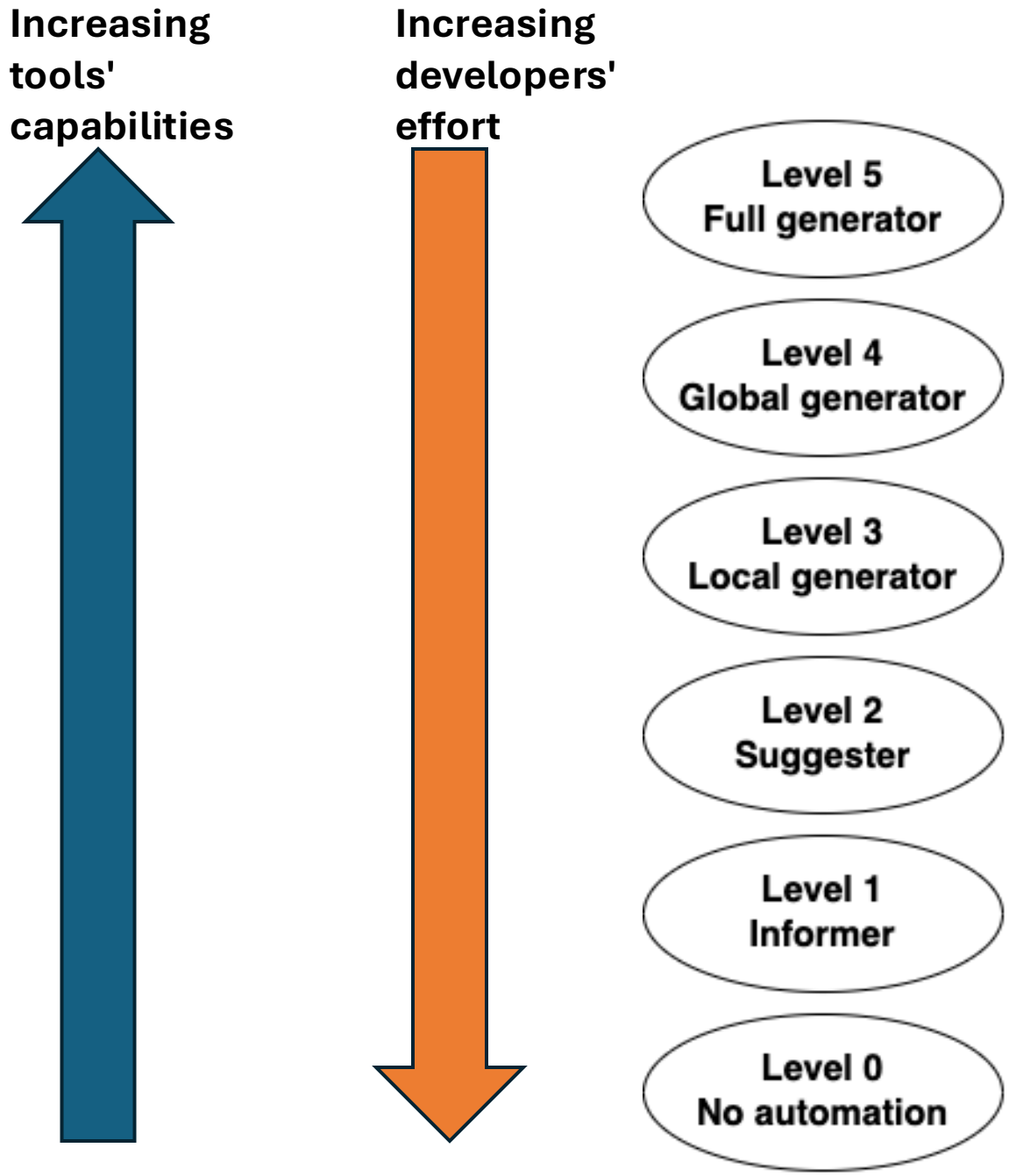
Keywords Software engineering automation · AI for software engineering · Generative AI

1 Introduction

Human history is defined by the use of tools. From the use of rudimentary stones for preparing food in the Paleolithic to rockets taking us to space today, tools have been used to augment our capabilities or even to give us newer and more complex ones. As with any other human activity, software engineering (SE) has also experienced, since its inception, the continuous creation of tools to increase productivity, reduce errors, and facilitate the work of those involved in the activity. High-level programming languages, integrated development environments (IDEs), and frameworks, just to name a few, allowed the development of increasingly complex software systems with reduced effort and time spent [28]. For at least a

DAnTE

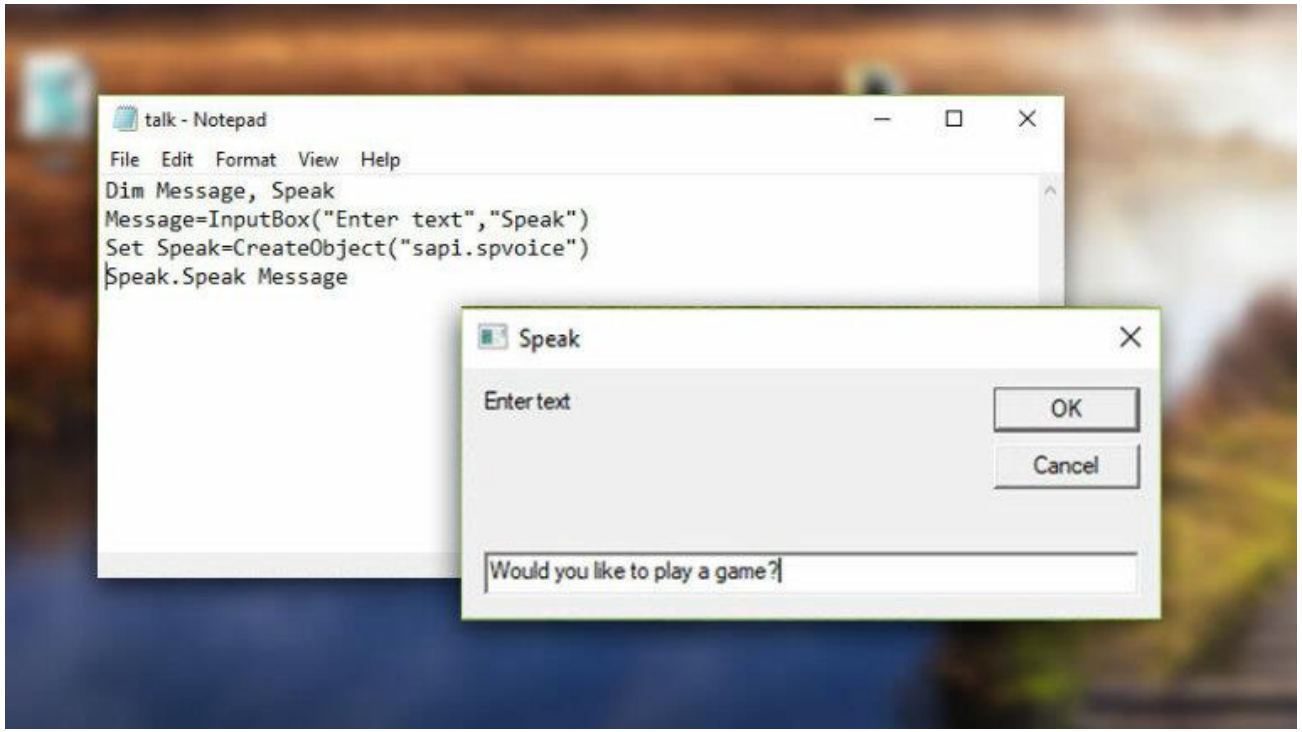
Degree of
Automation
Taxonomy for software
Engineering



Level 0 – No automation

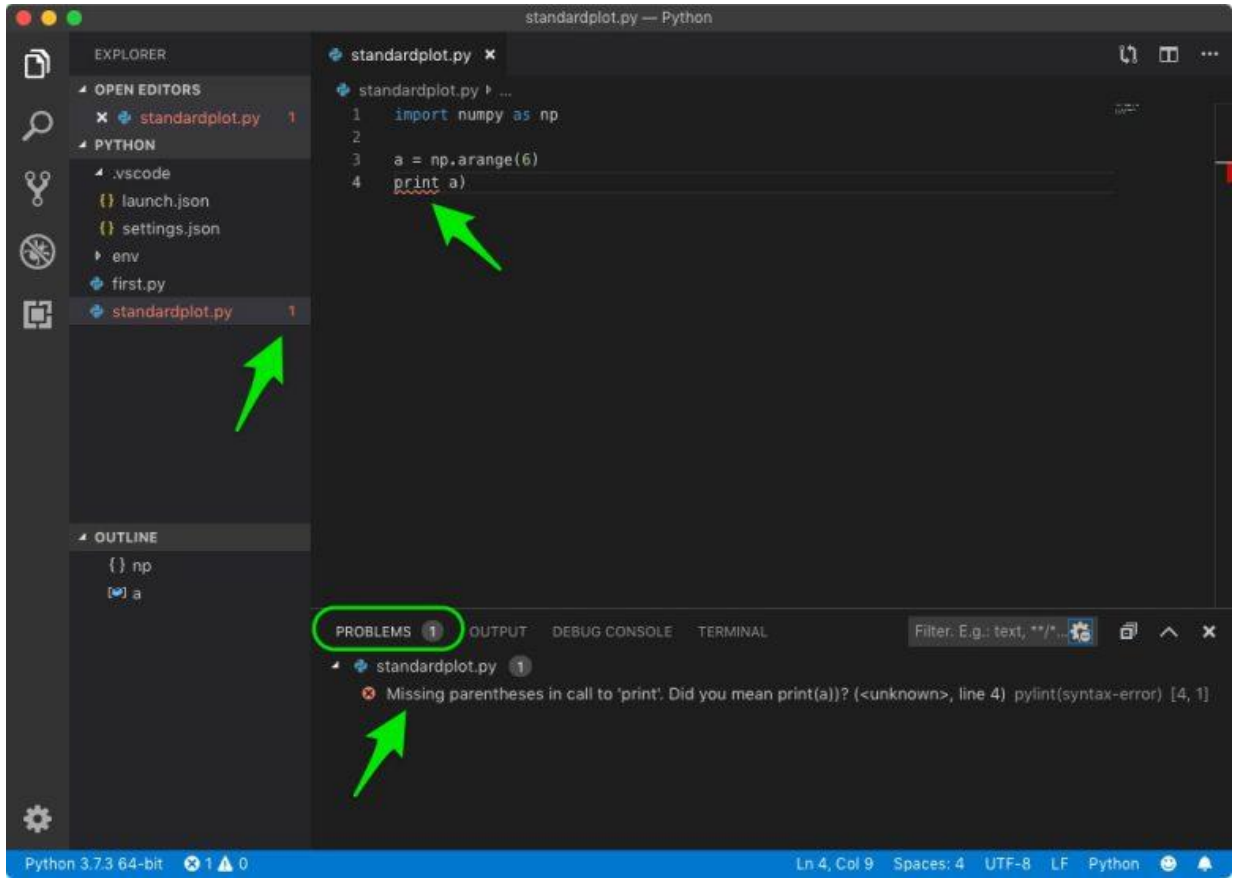
- Total lack of supporting tools

- Developers are responsible for all



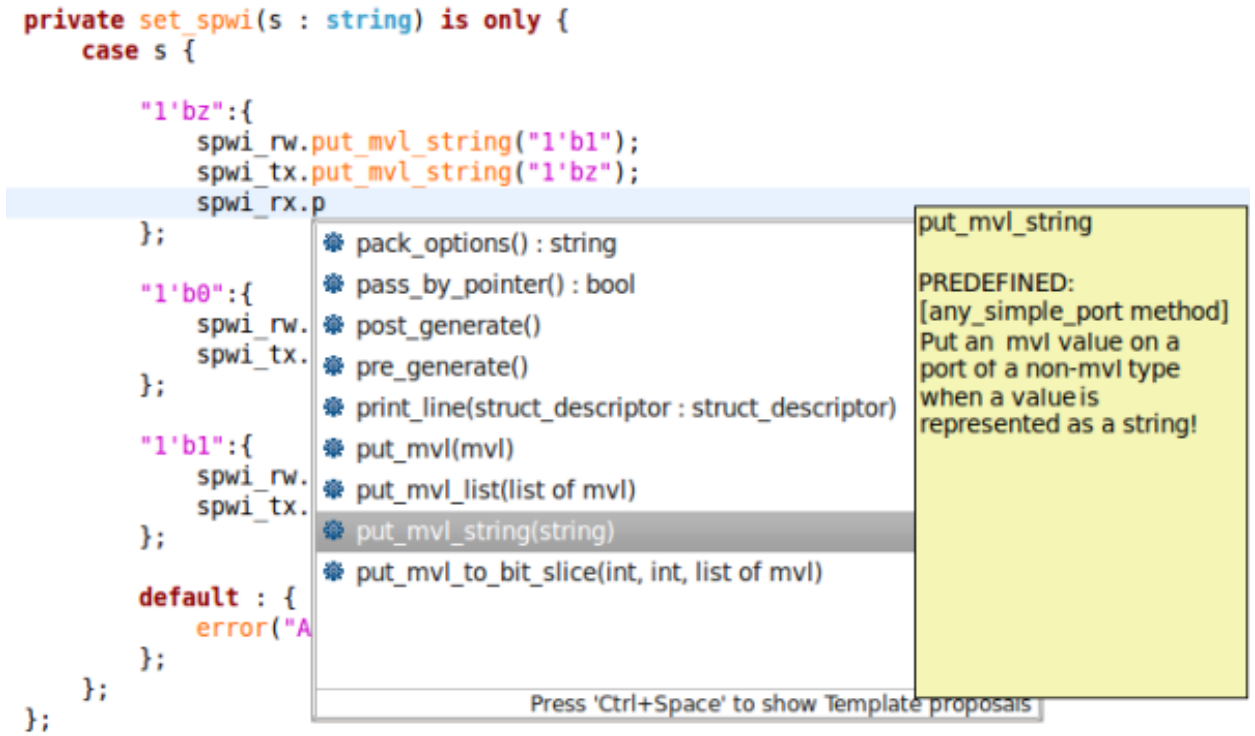
Level 1 – Informer

- Tools provide information but are not able to suggest solutions
- Developers are responsible for all



Level 2 – Suggester

- Tools can provide suggestions for modifying the implementation
- Developers perform the tasks but can accept suggestions from the tool



Level 3 – Local generator

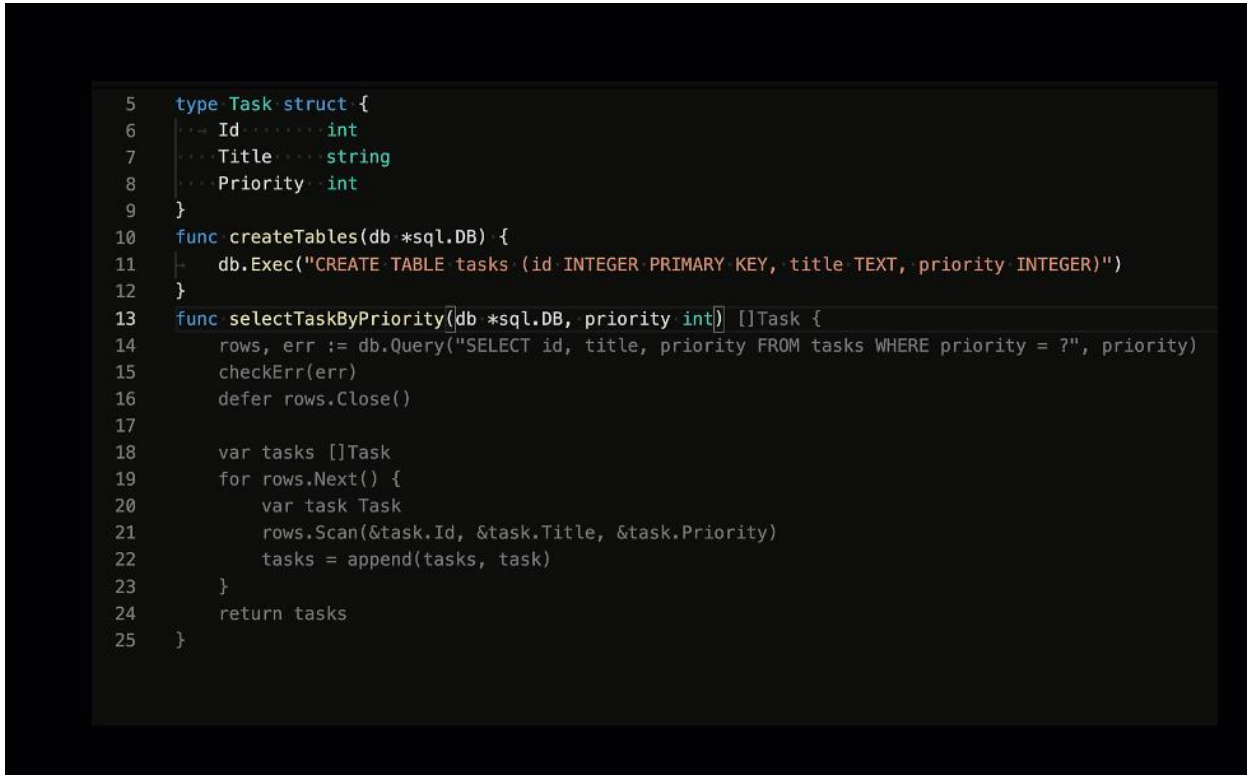
- Tools can **generate** solutions at **a restricted level** such as functions, methods, or classes

- At certain levels, developers can describe the needs and the tools produce one or more solutions that can be accepted or not

Level 3 – Local generator

- Tools can **generate** solutions at **a restricted level** such as functions, methods, or classes

- At certain levels, developers can describe the needs and the tools produce one or more solutions that can be accepted or not



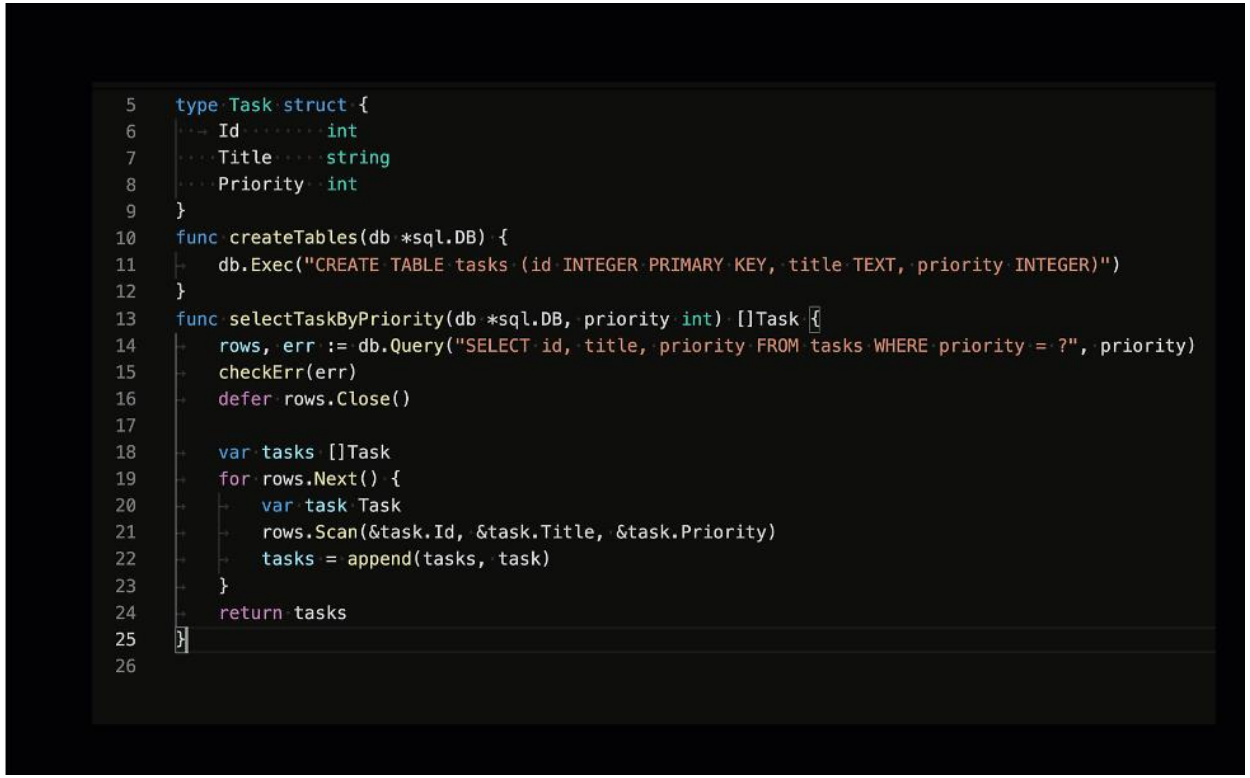
```go
type Task struct {
    Id       int
    Title    string
    Priority int
}
func createTables(db *sql.DB) {
    db.Exec("CREATE TABLE tasks (id INTEGER PRIMARY KEY, title TEXT, priority INTEGER)")
}
func selectTaskByPriority(db *sql.DB, priority int) []Task {
    rows, err := db.Query("SELECT id, title, priority FROM tasks WHERE priority = ?", priority)
    checkErr(err)
    defer rows.Close()

    var tasks []Task
    for rows.Next() {
        var task Task
        rows.Scan(&task.Id, &task.Title, &task.Priority)
        tasks = append(tasks, task)
    }
    return tasks
}
```

Level 3 – Local generator

- Tools can **generate** solutions at **a restricted level** such as functions, methods, or classes

- At certain levels, developers can describe the needs, and the tools produce one or more solutions that can be accepted or not



```go
type Task struct {
    Id       int
    Title    string
    Priority int
}
func createTables(db *sql.DB) {
    db.Exec("CREATE TABLE tasks (id INTEGER PRIMARY KEY, title TEXT, priority INTEGER)")
}
func selectTaskByPriority(db *sql.DB, priority int) []Task {
    rows, err := db.Query("SELECT id, title, priority FROM tasks WHERE priority = ?", priority)
    checkErr(err)
    defer rows.Close()

    var tasks []Task
    for rows.Next() {
        var task Task
        rows.Scan(&task.Id, &task.Title, &task.Priority)
        tasks = append(tasks, task)
    }
    return tasks
}
```

Level 4 – Global generator

- Tools can provide complete solutions, but they should be checked by developers

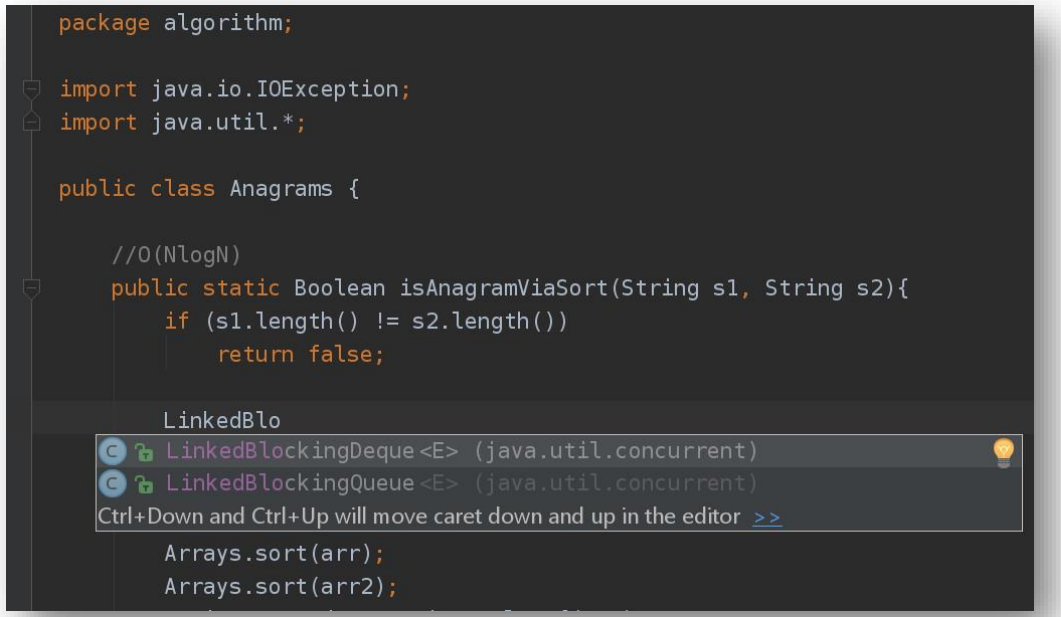
- Developers provide descriptions of the solution and check the proposed solutions.

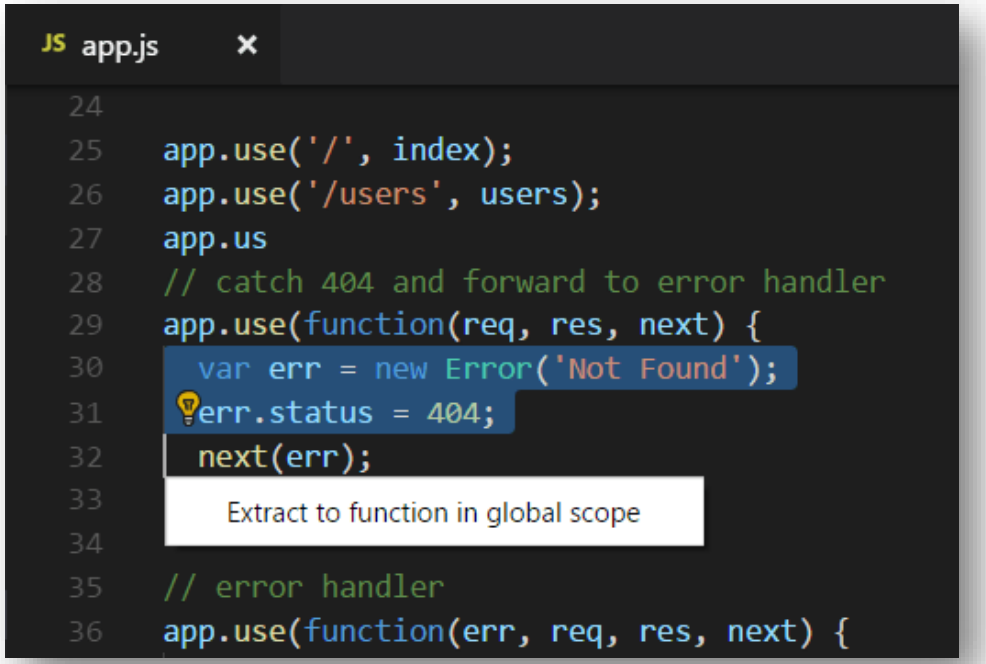
Level 5 – Full generator

- Tools can reliably create full solutions given descriptions
- Developers just describe what is expected

The current landscape

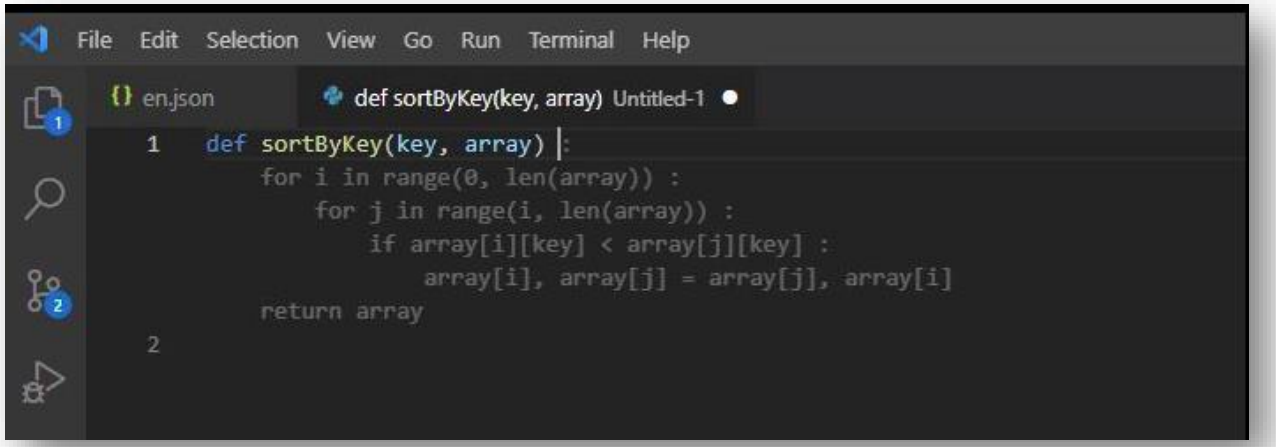
- Level 2 tools are consolidated and the de-facto standard





The current landscape

- Level 3 are trending



92% of programmers are using AI tools, says GitHub developer survey

AI isn't programming's future, it's its present.

Written by **Steven Vaughan-Nichols,** Senior Contributing Editor

on June 14, 2023

Reviewed by **Min Shin**

The current landscape

- Level 4 tools started being proposed

Towards Human-Bot Collaborative Software Architecting with ChatGPT

Aakash Ahmad
School of Computing and Communications, Lancaster University Leipzig, Leipzig, Germany
a.ahmad13@lancaster.ac.uk

Muhammad Waseem*
Faculty of Information Technology, University of Jyväskylä, Jyväskylä, Finland
mwaseem@jyu.fi

Peng Liang
School of Computer Science, Wuhan University, Wuhan, China
liangp@whu.edu.cn

Mahdi Fahmideh
School of Business, University of Southern Queensland, Queensland, Australia
mahdi.fahmideh@usq.edu.au

Mst Shamima Aktar
School of Computer Science, Wuhan University, Wuhan, China
shamima@whu.edu.cn

Tommi Mikkonen
Faculty of Information Technology, University of Jyväskylä, Jyväskylä, Finland
tommi.j.mikkonen@jyu.fi

ABSTRACT

...nplex process. ...ders' perspec- ...attern-driven ...ftware imple- ...ecture-centric ...of challenges. ...zed processes, ...expertise etc. ...ergent classes ...ained on large ...owledge with

KEYWORDS

Software Architecture, ChatGPT, Large Language Models, DevBots

ACM Reference Format:
Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Mst Shamima Aktar, and Tommi Mikkonen. 2023. Towards Human-Bot Collaborative Software Architecting with ChatGPT. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23), June 14–16, 2023, Oulu, Finland.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3593434.3593468

1 INTRODUCTION

Architecture of software-intensive systems enables architects to

Computer Science > Software Engineering

[Submitted on 11 Mar 2023]

ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design

Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, Douglas C. Schmidt

This paper presents prompt design techniques for software engineering, in the form of patterns, to solve common problems when using large language models (LLMs), such as ChatGPT to automate common software engineering activities, such as ensuring code is decoupled from third-party libraries and simulating a web application API before it is implemented. This paper provides two contributions to research on using LLMs for software engineering. First, it provides a catalog of patterns for software engineering that classifies patterns according to the types of problems they solve. Second, it explores several prompt patterns that have been applied to improve requirements elicitation, rapid prototyping, code quality, refactoring, and system design.

Level 5?

Utopia or future?

Thank you!