ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA CESENA CAMPUS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING - DISI

SECOND CYCLE DEGREE IN DIGITAL TRANSFORMATION MANAGEMENT

Class: LM-91

THESIS TITLE

Integrated Development Environments: Exploring the Impact of the Implementation of Artificial Intelligence on Workflow Efficiency and its Potential for Developer Displacement

Graduation thesis in
Software Engineering

Supervisor
Andrea Omicini

Candidate
Bruna Moema Dias Bahmed

Co-Supervisor
Giovanni Ciatto

Academic Year: 2023/2024
Session: First

Abstract

Starting from a comprehensive overview of Integrated Development Environments (IDEs), emphasizing their most important features and their tendency to evolve towards cloud-based solutions along with the advantages and disadvantages of that process, this thesis delves into the ramifications of incorporating Artificial Intelligence (AI) features within modern IDEs. With the recent shift towards the development of new AI-powered technologies and the improvement of existing ones, the incorporation of these technologies into IDEs is becoming progressively more prevalent. This research aims to provide a comprehensive examination of how the implementation of AI within IDEs can have an influence on the workflow of developers by streamlining development processes, and evaluates the potential risks associated with automation having a negative impact on traditional developer roles, eventually leading to the displacement of developers who operate in business settings. This thesis seeks to perform a thorough examination of these dynamics, offering insights into the characteristics and trends of existing IDEs, as well the opportunities and challenges presented by the integration of AI within them, thereby contributing to a deeper understanding of the evolving landscape of software development tools.

Contents

1 An Introduction to Integrated Development Environments

Integrated Development Environments (IDEs) are indispensable resources for software developers. They are all-encompassing software applications, equipped with all the necessary tools for software development. In this introductory chapter, we will be reviewing the technical features that IDEs have to offer, their historical journey, tracing their evolution alongside the ever-changing landscape of software development practices and technologies, as well as the some of the currently available options.
The final part of this chapter is instead dedicated to the Command Line Interface, which serves as a fundamental tool for interacting with a computer's operating system directly through text-based commands, providing users with precise control over system functions and allowing them to perform various tasks more efficiently, thereby also serving as a potential replacement for an IDE.

1.1 What is an Integrated Development Environment?

An Integrated Development Environment is not simply a code editor: while a code editor is fundamentally a text editor containing features that simplify writing code, an IDE is an all-inclusive software application equipped with an extensive range of tools and functionalities (one of those being a code editor) that are aimed at streamlining the entire software development process. Some of the main features of modern IDEs include:

- Code auto-completion, which consists of intelligent predictions of the code sequences while the code is being written [18]. Auto-completion can be achieved in several ways, such as performing statistical analysis of code sequences, using neural networks, Natural Language Processing techniques or deep learning models to perform the task [75]. IntelliSense is an example of a feature that, among other capabilities, enables auto-completion in Visual Studio Code (see Figure 1);
- Debugging, which consists in the location and adjustment of flaws in the code [18];
- Version control, which allows developers to track changes made to their files over time, and is particularly relevant when teams of developers are collaborating on the same project. Version control is usually accomplished through the integration with version control systems, such as Git and TFVC [18];
- Code refactoring, which enables developers to restructure their code as to make it more efficient and readable, without having to alter its behavior [18];
- A built-in compiler and/or interpreter, which directly converts the human-readable source code into machine-readable code;
- Collaborative editing features, which can come in very handy in team projects, where more developers work together on the same project;
- Deployment features, which enable developers to make their applications available to its target users;
- Overall project organization, by allowing developers to organize their files in a structured format [18].

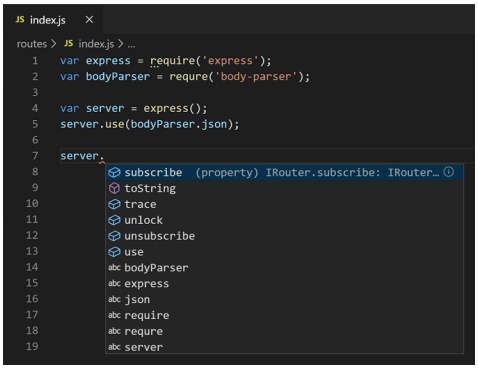

Figure 1: IntelliSense feature in Visual Studio Code [81].

Certain IDEs may also include plugins and/or extensions, which are software tools designed to enhance the IDE's capabilities by introducing additional functionalities aimed at performing specific tasks.

With the increase in the complexity of software projects and the overall quick evolution of software technologies, IDEs have emerged as essential tools for developers across different business sectors worldwide. In fact, we have reached a point where IDEs represent not just a set of development tools, but a cornerstone of modern software engineering, which allow developers to accurately and efficiently turn their project ideas into reality.

Although IDEs are more feature-rich than code editors, that does in no way imply that editors are no longer meant to be used. One of the most well-known and highly regarded software applications for programming is Visual Studio Code by Microsoft, which is in fact considered to be a source code editor, and not an IDE. It offers a comprehensive environment suitable for users at all skill levels, from beginners to experienced programmers, while also supporting all of the mainstream programming languages.

VS Code does not necessarily have to be downloaded either: thanks to Visual Studio Code for the Web (see Figure 2), it can also run entirely on a browser, which is very convenient as it allows developers to work on the same projects across different devices.

It also has a dedicated marketplace for its supported extensions, which range anywhere from theme customization to support for new programming languages. Although VS Code is not typically classified as an IDE, the diverse array of extensions it offers can make it an extremely feature-rich editor, thereby making it almost comparable to one [80].

The equivalent of Visual Studio Code in the shape of an actual IDE is instead Visual Studio by Microsoft, a well-established, widely used and acclaimed IDE, which offers the most comprehensive support for .NET and C++ development on Windows at the moment [79].

Conversely, an example of a source code editor that is much less well-equipped than VS Code is Notepad++ (see Figure 3), designed to be a programming-friendly alternative to Windows Notepad. It contains features that aid in code composition, such as auto-completion and syntax highlighting, with the latter consisting of displaying code in different colors.

Most code editors will suffice for novice programmers still learning how to code, or for handling projects and programming tasks at a very small scale. Instead, for more complex projects that require advanced features, opting for an IDE is usually the wisest choice. Another example of an esteemed, well established IDE is Eclipse IDE (see Figure 4), created by Eclipse Foundation, a non-profit organization that supports open source projects in computer science [64].

In fact, Eclipse IDE is a completely free and open-source tool. Although it does not support as many programming languages as for instance Visual Studio Code does, given that it is primarily used for developing Java applications, it also supports other commonly used programming languages such as C and C++, Python, PHP and JavaScript, making it suitable for the development of several different types of applications.

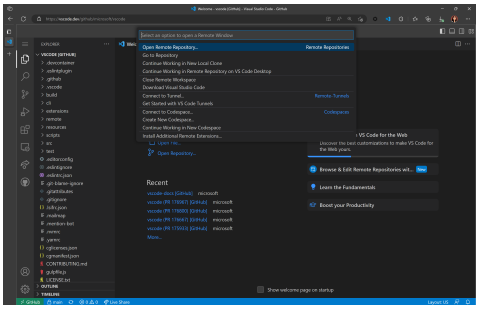

Figure 2: VS Code for the Web's user interface [80].

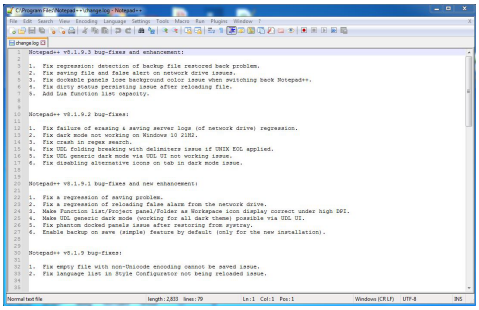

Figure 3: Notepad++'s user interface [34].

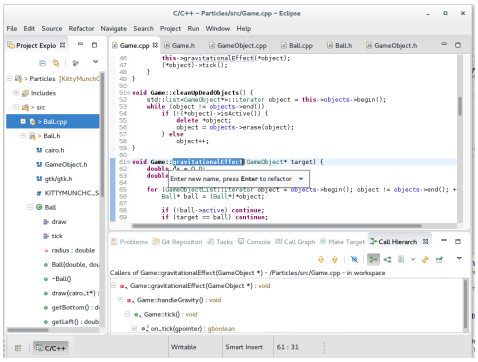

Figure 4: Eclipse IDE's user interface [64].

Integrated Development Environments offer a comprehensive platform for software development, combining essential tools and features tailored to each step of the coding process. By providing a unified interface for coding, debugging, and project management, IDEs enhance productivity and efficiency for developers of all levels. Each solution brings its own unique set of features and advantages, catering to different programming languages, project requirements, and personal preferences. It is therefore always crucial to carefully consider all these factors before deciding on the IDE to use in a specific circumstance.

1.2 Origins, History and Evolution

It is hard to imagine a time where writing software was even more complex than it is now. Up until the late 1970s, the software development process was much less streamlined than what most of us are used to: Integrated Development Environments were yet to come into existence, which meant that developers had to use independent tools for each step of the software development process, leading to a fragmented and much more time-consuming software development workflow. In fact, prior to the existence of IDEs, it was necessary to actually assemble and integrate the development environment.

The purpose of the first IDEs was therefore to integrate the different phases that comprise the lifecycle of a software product. Turbo Pascal (see Figure 5) was one of the first established IDEs, introduced in 1983. It was an all-in-one solution made to be used for the Pascal programming language, which combined a code editor with a debugger and a compiler [24].
The very first IDE is instead considered to have been a German project under the name Program development system terminal (PET), whose first prototype was developed in 1975 by Softlab Munich. It was later renamed as Maestro I, whose development was co-founded by the German government, with the purpose of creating an interactive programming terminal that would cost around 1.000 Mark per month [27].
Prior to the invention of Maestro I, programmers would input their code and test data onto paper tape or punched cards. Once the punching was completed, the programmer would then insert the tape and/or cards into the computer for processing, which naturally came to be a very burdensome process. The introduction of Maestro therefore caused an extremely impactful shift in the lives of developers at the time [27].
Following its first presentation in 1975, in 1977, Maestro I was connected to Mainframe computers, which are high-performance computers designed to handle large-scale tasks.
About a year later, it was exported to the United States, where Maestro's creator company Softlab founded another branch. Maestro's very first customer was the Boeing Company, and its biggest purchaser eventually became the Bank of America [27].
Maestro I's software was a proprietary Four-Phase Disc Operating System, which was comprised of a text editor, an assembler, some compilers and a linkage editor [27]. Its basic hardware system was instead described as a "key-to-disk" data entry system, an obsolete data system in which the inputed data was accumulated on a magnetic disk before it was verified [30].

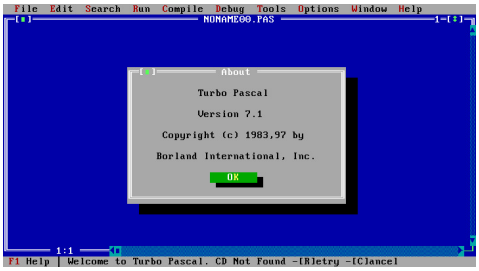

Figure 5: Turbo Pascal's user interface in 1983 [42].

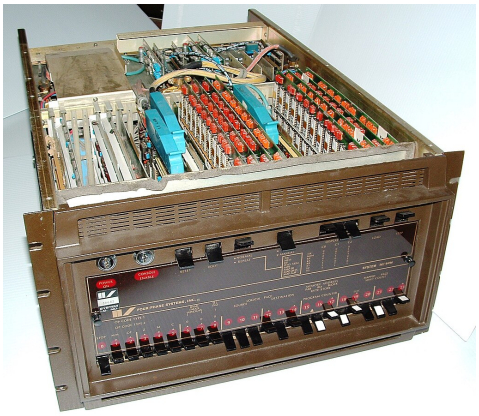

Figure 6: Maestro I's Central Processing Unit (CPU) [92].

Ever since the release of these two pioneer IDEs, and especially following the advent of the Internet, technology has evolved at an unprecedented speed. As Windows gained broader adoption during the 1990s, Visual Studio made its debut in 1997, under the name Visual Studio 97 (see Figure 7). It included all existing Microsoft development tools at the time (Microsoft Visual Basic, C++, J++, FoxPro, InterDev and Microsoft Developer Network Library), and delivered comprehensive support for programming languages, reusable components, features aimed at scalability (which included the support for team-based development,

executables with access to enterprise data and transaction support) and open standards, such as Java and HTML. These were defined as "key developer needs" by the Microsoft community at the time of Visual Studio's first release [19].
The new release was aimed at enabling three specific scenarios, which were to extend client/server to multitier architectures, activate the web by building Web applications and server-side databases, and integrate the Internet through the implementation of Web interfaces to already existing client/server and legacy systems at the time [19].
The 1990s marked the advent of widespread public access to the Internet, a milestone that profoundly influenced the direction of technological innovation for the years to come. It is evident that, in response to this turning point in the computer science landscape, Visual Studio 97 decisively shifted the focus of its offerings towards the integration with the Web.
In the current technological landscape instead, as cloud solutions gain prominence, IDEs are increasingly pivoting towards integration with the cloud. An excellent example of this new approach is JetBrains Gateway by JetBrains, a renowned software development company known for crafting esteemed IDEs such as IntelliJ IDEA.
JetBrains Gateway comes in the form of a desktop app that allows users to remotely access the IDEs made by JetBrains, without having to install them in their machines. This solution entails running the backend of those IDEs on remote environments hosted in the cloud [45]. Solutions like JetBrains' come with the enormous advantage of not having to store any code in one's machine, meaning that the same codespace can be accessed from different machines in different locations.
Replit, a cloud IDE, also serves as an example of this trend. It consists of a web-based code editor which supports all of the most commonly used programming languages, as well as real-time collaboration between different users. Each user's projects can also be publicly shared, which allows them to be readily accessible to fellow Replit developers. In fact, at the moment, private projects are only available for users with a paid plan, which limits users in the free tier to creating public projects only [67].
Recent advancements in the fields of Data Science and Machine Learning have fostered a widespread adoption of the Python programming language. Since then, several IDEs and web applications have been created to meet the growing demand for tools tailored to Data Science and Machine Learning tasks, such as PyCharm by JetBrains, which besides being ideal for Data Science and Machine Learning endeavours, is also suitable for web development projects with Python [46].
Project Jupyter instead exemplifies a project conceived to foster interactive Data Science and scientific computing. The Project Jupyter team brought JupyterLab (see Figure 8) into existence, which consists of a web-based IDE tailored to Data Science and Machine Learning projects [47].
The evolution of IDEs has therefore been a long journey marked by remarkable advancements in landscape of software engineering practices. Beginning with early solutions that required programmers to code using tape and punched cards, IDEs have progressed to modern, feature-rich development environments that enable programmers to navigate the entire project lifecycle either by using their own physical machines or remotely with cloud resources, from ideation all the way to deployment.
Throughout their evolution, IDEs have continuously adapted to meet the evolving needs of developers as technology achieved its milestones. They have always strived to enhance productivity and optimize the software development process, and it is reasonable to anticipate that they will continue along this path for the foreseeable future.

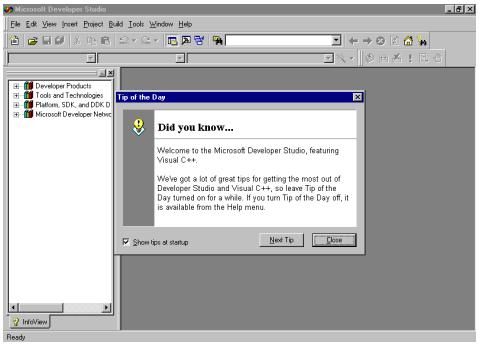

Figure 7: Visual Studio 97's user interface [7].

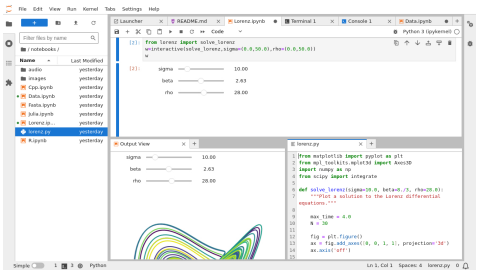

Figure 8: JupyterLab's user interface [48].

Considering the recent pivot towards Artificial Intelligence (AI) - powered solutions, and the unfolding potential of Large Language Models (LLMs), it is interesting to contemplate the potential expansions IDEs might incur in the coming years.

1.3 The Command Line Interface

The Command Line Interface (CLI), often also referred to as the Terminal or the Shell, is a very basic interface where users input commands, which are lines of text that adhere to a predefined syntax, and receive a response from the command, which, naturally, is also in the form of text. The interaction with the CLI therefore happens almost entirely through the keyboard [71].

In the software development community, a common assertion is that, the more experienced and especially passionate programmers will rely solely on their computer's CLI to complete tasks of various types. For adept developers, mastering the CLI can potentially eliminate the need for an IDE (which falls under the category of GUI applications). Indeed, the kinds of tasks that can be performed on the CLI just as on the Graphic User Interface (GUI) range anywhere from navigating and interacting with the filesystem and manipulating files, to invoking version control systems and package installations, and dealing with infrastructure [73].

Although that statement may well be true, it is important to ask ourselves why someone would choose to use a Command Line Interface, instead of a Graphic User Interface, which is a much more complex and interactive interface where users will communicate with their computers not through text prompts, but through clicks and selections.

One of the main reasons one may opt for using the CLI instead of the GUI is the speed at which one can perform the same tasks. Once someone has attained a certain level of proficiency in utilizing the CLI, it is arguably easier and faster to simply prompt a couple of commands into it than it is to navigate through the user interface to accomplish the same tasks.

For instance, by utilizing the Command Prompt on a Windows PC, users can easily navigate to a directory of their choice by entering the command *cd* followed by the directory's path (for example, if the user is currently in the root directory, *cd Downloads* will get them to the Downloads directory), after which they can get a list of the files and folders contained in the current directory by typing *dir* (which in this case would get them the list of downloaded contents in their computer).

Of course, it is possible to do a lot more than accessing directories and their contents on the

CLI. The advantage of using the CLI with respect to the GUI is most evident when it comes to repeating simple tasks across multiple files. The complication with that is it can be impossible to undo changes made through the CLI, which means that it is a tool that ought to be used with caution [65].
For manipulating and editing files directly from the CLI, it is necessary to install and Command Line Editor (also referred to as Terminal Text Editor). The Command Line Editor allows users to create, modify and save files directly from the CLI, eliminating the need for an external GUI application, such as an IDE.
Some examples of commonly used Command Line Editors are Vim (Figure 9) and Nano for Linux and macOS systems. With Vim being reported as more complex to use than Nano, it makes for a more suitable choice for experienced CLI users, while Nano is ideal for novice developers [34]. Windows systems instead do not offer a built-in Command Line Editor, meaning that Windows users have to either use the Notepad (which is the default text editor in Windows systems), or install a clone of an editor made for another operating system.

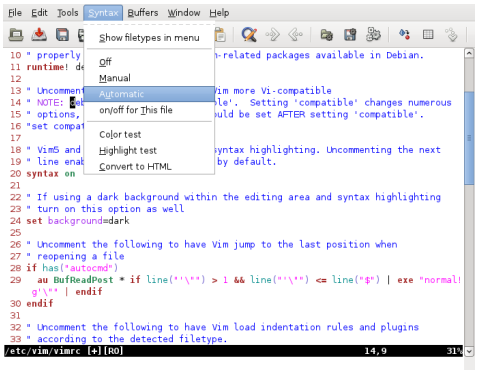

Figure 9: Vim Editor, a Command Line Editor for Linux [69].

The main reason why a user may prefer the GUI over the CLI, is the fact that they would have to memorize the commands to perform specific tasks, while when using a GUI everything is much more intuitive, and there is no need for remembering specific text commands. For exactly this reason, it is common for CLI users to customize their command-line by creating shell *aliases* for frequently used commands, or arrangements of commands.
An alias replaces a given name, the alias, with a string that defines a conventionally complex command, or chain of commands. Many users publicly share these aliases on collaborative social coding platforms such as GitHub (which is at the moment the most popular platform for managing and sharing code), thereby contributing to a collective knowledge of command-line customizations. This can provide insight into the tasks that expert programmers perform on a daily basis, and on how well the standard Command Line environment supports such tasks. [73].
The following is an example of this practice: *alias ll = 'ls -l'*. In this expression, *ll* is the alias name, and *ls -l* (a command aimed at listing the contents of a directory, and the argument *-l* specifying the output format as long-form) is the alias value. As previously mentioned, the alias can also be used to replace a chain of commands, which can be fulfilled by separating those commands with a pipe, which is a vertical slash [73].
In their research paper on Command Line Customization, written with the purpose of studying how the standard environment of the CLI could be expanded, enhanced and modified in a constructive manner for its users [73], Schröder and Cito identify nine most common

customization practices. These practices are further distributed by the authors among three different kinds of aliases, those being: *Shortcuts* (giving a long, complex expression a shorter name), *Modifications* (changing the meaning of certain commands) and *Scripts* (combining two or more commands, usually with the purpose of altering data or binding subcommands):

Shortcuts	Modifications	Scripts
Nicknaming Commands	Substituting Commands	Transforming Data
Abbreviating Subcommands	Overriding Defaults	Chaining Subcommands
Bookmarking Locations	Colorizing Output	
	Elevating Privilege	

Table 1: Customization practices for commands in the Command Line Interface [73].

- *Nicknaming commands* consists of simply giving a new name to a command, without adding any arguments;
- *Abbreviating subcommands* means simply abbreviating subcommands, which are keywords that can be attached to commands with the aim of unlocking additional features, such as the argument *init* in the expression *git init*;
- *Bookmarking Locations* are situations in which the alias acts as a bookmark to a location (such as a file path) referenced by the argument with which a specific command is called;
- *Substituting Commands* define situations where, due to an alias name being the same as the name of a pre-existing command, the alias defines a substitution for that command;
- *Overriding Defaults* means that the alias re-defines the command whose name is the same of the alias by overriding its default settings;
- *Colorizing Output* simply consists of applying color to elements displayed in the CLI;
- *Elevating Privilege* by inputing the *sudo* command, which allows the user to execute another specified command with superuser privileges;
- *Transforming Data* by using the pipe, which creates an interface between two otherwise separate programs;
- *Chaining Subcommands* by running multiple subcommands in succession, for instance *brew update && brew upgrade*, where *brew update* updates the package database, and *brew upgrade* upgrades the previously installed packages to the most recent versions available.

Command aliases are a widespread practice among command line users. In fact, Schröder and Cito have identified well over 2 million shell alias, all collected from GitHub [73]. Aliases can therefore significantly enhance productivity by reducing typing effort and simplifying intricate commands into memorable shortcuts. By leveraging aliases, users can navigate the command line with more ease and greater efficiency, thereby saving time and effort in their daily tasks.

Although the CLI can supplement several aspects of an IDE, its ability to fully replace an IDE varies based on project requirements and developer preferences. While it is feasible to write most programs using just a CLI (or text) editor and the CLI itself, some users may find greater ease in employing an IDE for that purpose. The choice between the CLI and an IDE is therefore highly subjective, and it is completely dependent user preferences and proficiency levels in either tool.

2 Cloud Integrated Development Environments

The idea of not limiting one's development workspace to a single machine can be very appealing. Cloud-based Integrated Development Environments were created with the purpose of fulfilling exactly that requirement. In a nutshell, cloud IDEs enable the development environment to run on remote servers, while the IDE's interface itself can be accessed either locally, or more typically on the Web through a browser window [9].
In this chapter, we will delve into the emerging landscape of cloud computing technologies, and more specifically, of IDEs based on the cloud. We will be discussing the currently available options, and the added value that cloud IDEs can bring with respect to their locally installed counterparts, both for independent developers and companies, as well as their drawbacks and potential issues that both individuals and companies using cloud IDEs ought to be aware of.

2.1 A Brief Introduction to Cloud Computing

Cloud IDEs are a subset within the broader domain of cloud computing, which can be defined as the on-demand availability of computing resources through the Internet. Cloud computing technologies make it unnecessary for individuals and/or companies to autonomously manage physical computing resources, thereby enabling them to pay exclusively for what they actually use [14].
In exchange for a fee, individual users and/or businesses can subscribe to unlock access to a virtual pool of shared resources, comprising computing, storage, and networking amenities, all of which are hosted on remote servers that are owned and overseen by the cloud service providers [14].
Cloud computing deployment models are split into three categories, which are the following:

1. Public clouds, which are managed by third-party cloud service providers, and allow companies to gain access to on-demand resources (such as network, compute and storage) according to their specific requirements and purposes [14].
   The third-party service provider is responsible for managing all of the necessary hardware and software in a network of data centers distributed around the world, which allows companies to easily get access to the latest technologies in fields such as artificial intelligence (AI), Internet of Things (IoT) and blockchain [4]. A few of the major public cloud providers are Amazon Web Services (AWS), Microsoft Azure and Google Cloud;

2. Private clouds, which are instead managed by a single organization and hosted privately in that organization's own data centers. Private clouds therefore provide companies

with greater control and security of data, while also enabling access to on-demand resources [14];

3. Hybrid clouds, which merge public and private cloud models by integrating the company's internal resources with services and infrastructure managed by third-party cloud service providers [4]. This allows companies to maintain the security levels associated with private clouds, while also leveraging public cloud offerings [14].

Cloud computing service models are also split into three main offerings (see Figure 10):

1. Infrastructure as a Service (IaaS), a service which offers on-demand access to technology infrastructure, which may include servers, storage, network and virtualization [14];

2. Platform as a Service (PaaS), which provides individuals and companies with access to hardware and/or software resources necessary for cloud application development [14]. PaaS clouds are particularly interesting solutions for software engineers, given that they allow developers to focus exclusively on developing software applications. When employing PaaS solutions, users are not required to take care of operational tasks, and the cloud itself handles the varying workload via auto-scaling, a feature referred to as *elasticity* [13].

3. Software as a Service (SaaS), which provides the full application stack, from its infrastructure to the software itself, and usually comes in the form of an end-user application [14].

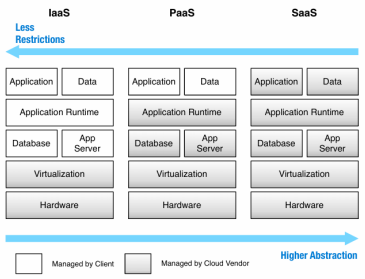

Figure 10: Resource management modalities in cloud computing models [13]. IaaS models require the most management by the client, while SaaS models require the least.

The main advantage that companies get when it comes to employing cloud computing technologies instead of on-premises solutions is that they are able to access scalable resources without the need to worry about overspending, or resource shortages [14].
For that reason, organizations with extremely high business growth that exceeds current infrastructure constraints, or, on the contrary, insufficient utilization of existing infrastructure resources, are considered to be ideal candidates for cloud computing solutions [14].
Cloud computing finds versatile application across various types of operations commonly undertaken within companies. The most intuitive application of cloud computing technologies in business settings is infrastructure scaling, as a response to companies' changing requirements in terms of compute capacity [14].

Other typical applications encompass areas such as data storage, big data analytics (the processing of extremely large volumes of data), disaster recovery (companies can leverage cloud computing technologies to back up their digital assets in case of emergency situations, such as cyber attacks, natural disasters or business disruptions of any kind) and application development [14].
Besides traditional enterprise activities, cloud computing technologies have expanded to various domains that are not necessarily within the realm of IT. Several kinds of services that were previously reliant on local installation are now also accessible through the cloud. We can consider video games as one of the prime examples of that trend: with offerings like GeForce Now, Nvidia's cloud gaming service, users can play all sorts of video games entirely online.
Office productivity software applications, such as Microsoft Office, have also evolved into cloud-based services like Microsoft 365, which was previously Office 365.
Cloud computing technologies can also be applied in the realm of finance and banking. Examples of applications in those fields are mobile retail trading (through which retail investors can execute transactions using a mobile device) and widget-based Internet banking applications (which are applications whose users are able to customise their Internet banking interface by adding, removing or reordering widgets; thereby allowing them to create a personalized banking experience that is tailored to their specific needs) [90].
Applications can now also be built and ran entirely within the cloud, thanks to Cloud Native, which is a new software approach of building, managing and deploying software applications in cloud-based environments [5].
Unlike monolithic software applications, which are applications whose components are combined into a single unit, cloud-native applications are decomposed into loosely coupled services, which contributes to an improvement in speed and scalability of software delivery [91].
The pillars of cloud-native development are:

- Microservices, which imply splitting an application into a group of lightweight services that are able to interact with each other via Application Programming Interfaces (APIs) [91];
- Containers, which are lightweight components containing all of the elements necessary in order to run code in any environment [91];
- Continuous Integration (CI), in which application changes are frequently and faultlessly integrated within a shared code base, with the purpose of identifying and solving potential issues more efficiently [5];
- Continuous Delivery (CD), meaning that code is automatically deployed into the cloud. CI and CD are combined in cloud-native development for fast and efficient delivery of applications [5] ;
- DevOps, a software development model aimed at optimizing the collaboration between development and operations teams [5];
- Serverless, a cloud-native model in which the application's underlying server infrastructure is entirely managed by the cloud service provider [5].

Cloud-native applications benefit from the scalability and flexibility which both allow them to be quickly updated to meet ever-changing customer demands [5].

Among the advantages of cloud-native applications with respect to regular applications are quicker development (since feature updates do not cause downtime), consistency and reliability when it comes to the operating environment, and cost efficiency (since the application owners are only required to pay for the amount of resources that the application actually uses) [5].
Embracing Cloud Native as an application development modality therefore empowers organizations to innovate faster, reduce expenses and provide their clients with highly available applications [5].
In conclusion, cloud computing has revolutionized how organizations purchase, scale, and manage computing resources. By taking advantage of cloud computing technologies, companies can gradually scale their computing resources as they grow as entities.
The different available deployment (public, private and hybrid) and service (IaaS, PaaS and SaaS) models offer computing environments tailored to meet diverse personal and organizational needs. Applications that were previously meant to be used locally, such as video games and office productivity software, are also slowly migrating to the cloud.
Finally, thanks to Cloud Native, scalable applications can now be developed and managed entirely on the cloud.

2.2 Software Development Brought into the Cloud

The traditional concept of an Integrated Development Environment considers a locally installed application. In essence, an IDE will be installed on a specific machine, and that machine is where the source code written within the IDE will reside.
Cloud IDEs, on the other hand, are hosted entirely on the cloud by leveraging resources from a network of virtual and/or physical servers, which provides developers with the flexibility to access their projects from any machine in the world. This sort of freedom is only destined to grow in its importance, considering the increasing tendency towards remote work [76].
The shift towards cloud IDEs can be viewed as somewhat of a consequence of a broader trend under the name of Infrastructure as Code (IaC) [58], which can be defined as the provision and management of infrastructure through code, instead of manual hardware configuration. Thanks to IaC, infrastructure can be supplied through the execution of a script [60].
Traditionally, each developer was responsible for their own environment setup. However, a notable consequence of developing and testing code in one environment and deploying it to another is that, occasionally, what works on the former machine will not necessarily work on the latter [58]. For that reason, companies have reached the conclusion that the local development model needed to be brought into the cloud, and that is how cloud IDEs came into existence [58].
As we have seen, cloud development enables quick scalability [15], given that cloud resources are essentially unlimited [8]. This aspect of cloud computing technologies is particularly important when it comes to undertaking resource-intensive tasks. In fact, when using a locally installed IDE, the tasks that can be performed are restricted by the computational capacity of the machine where the IDE is installed.
In contrast, when using a cloud IDE, owning a powerful computer is not a primary necessity, because the IDE will rely on cloud resources to handle computationally intensive tasks. Rather than relying on the local machine's CPU to run code, the computing and processing tasks are handled by remote servers [58]. This aspect is especially beneficial to companies, as they are no longer required to provide expensive, potent hardware to their employees, which

may result in significant cost savings for the businesses making use of this technology [9].
Another reason why the exploitation of remote resources are a major advantage for companies is it implies that, when opting for cloud development solutions, the need for upfront planning is significantly reduced [78]. Companies therefore do not have to purchase all of the computing resources for the projected growth of the company's computing needs at the very start of operations. Instead, they can gradually scale cloud resources in alignment with the evolution of their processing requirements, which once again, inevitably leads to cost reductions for the company [78].
Since the development workspace can be accessed remotely, cloud IDEs also facilitate collaboration between multiple developers within the same project [9], which is something that usually can only be achieved on locally installed IDEs through additional plugins and/or extensions that provide real-time collaboration features.
Another significant benefit that comes with using a cloud IDE is that the constraint of aligning the machine's operating system with that for which the IDE was designed completely disappears. In fact, while some locally installed IDEs are compatible only with a specific operating system, cloud IDEs can be used irrespective of the operating system installed on the user's machine [76].
When it comes to the pricing models for cloud IDE solutions, most of them offer different pricing plans which may vary based on the available features, number of workspaces or hours of usage.
Most cloud IDE providers offer an Enterprise plan, which is tailored for companies and usually comes with exclusive features, such as dedicated support, the ability of choosing the cloud region, on-premise (which means that the organization can use its own servers and infrastructure) deployment, custom Service-level Agreements (abbreviated as SLAs, which are agreements between customers and service providers that define the expected quality of the service by the customer), single sign on (which allows users to access multiple services and tools with a single set of login credentials) [17] and sole-tenancy solutions (meaning that companies can have access to an infrastructure that is dedicated to hosting their projects only) [67].
Considering the growing trend in the technological arena towards cloud-based solutions, along with all of the previously mentioned benefits of transitioning development environments to the cloud, several alternatives for cloud IDEs have emerged in the last few years. As stated earlier, cloud IDEs are made so that, while the development environment runs on remote servers, the editor can be accessed either locally, or through a browser.
Different modalities give rise to a cloud IDE, which are either the creation of a completely new IDE hosted exclusively in the cloud, or the transition of an existing locally installable IDE into the cloud.
In Chapter 1, we had the chance to take a look at a couple of examples of cloud-based development environments, specifically Replit and JetBrains Gateway. Replit (see Figure 11) is a more classical example of a cloud IDE, in the sense that it is accessible entirely through a browser window, thereby making any local installation completely unnecessary. It was born as a cloud IDE, meaning that there was never a version of Replit that could be installed locally.
As previously mentioned in Chapter 1, by default, the projects, which are referred as *Repls* within the platform, are public and can be accessed by fellow Replit users. Projects can also be shared with specific users, which enables live collaboration. Although collaborative editing is supported within Replit, the platform currently does not provide a plan tailored for

teams and/or enterprises, which makes it especially suitable for solo developers, rather than teams of developers.
When it comes to creating new Repls, it is possible to either build a project from scratch, or import an existing one from GitHub. Replit is also known for having a highly intuitive interface, which makes it suitable for programmers of all levels of expertise.
The platform has recently also integrated in-IDE AI assistance, under the name of Replit AI, which is at the moment available to all Replit developers, including those in the free tier. The primary focus and selling point of Replit AI currently lies in automating the repetitive aspects of coding [68].
Another very complete and convenient solution within the landscape of browser-based cloud IDEs is Codeanywhere (see Figure 12), which just like Replit, was always cloud-based, and consists of a web-based editor which supports all of the major programming languages. Codeanywhere also comes with a web-based terminal, and enables its users to connect their own servers and set up containers that are preconfigured for the programming environment of their choice. Like most cloud IDEs, Codeanywhere also provides real-time sharing of projects and live collaboration features [17].

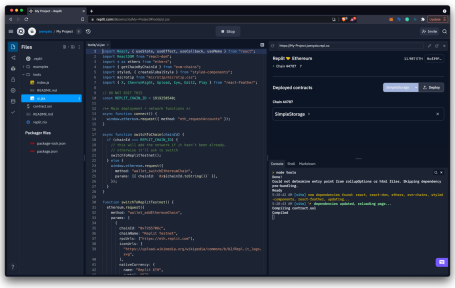

Figure 11: Replit's user interface [25].

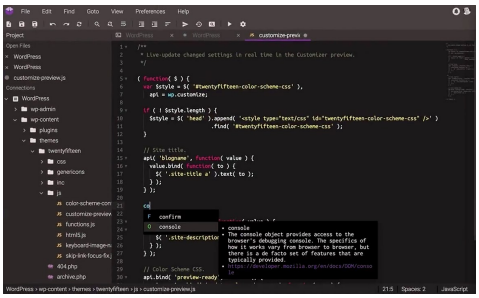

Figure 12: Codeanywhere's user interface [1].

JetBrains Gateway (see Figure 13), on the other hand, as previously mentioned in Chapter 1, is a different type of solution, consisting of an application that provides users with remote access to the existing IDEs made by JetBrains.
JetBrains Gateway is quite an interesting solution in its uniqueness, because while the Gateway itself has to be installed on the user's machine, the IDE selected by the user (which can be any of the existing IDEs by JetBrains) runs remotely on the cloud. The IDE is set up as a backend service remotely, while a thin client running locally connects to the backend of said IDE, and loads its user interface [28]. In short, while the IDE's GUI is accessible locally within the user's machine, the backend of the workspace is hosted remotely on the cloud.
Another aspect of JetBrains Gateway that makes it a convenient choice for developers and companies is that the backend of the environment can run on several different providers (see Figure 13), among which Google Cloud, Daytona, Gitpod, Amazon CodeCatalyst, Coder, or any machine via SSH [45]. Therefore, the user has the freedom to select both the IDE and the provider where its backend will run, according to their preference and to what best suits their project requirements, and is therefore never limited to a singular solution.
The IDE remains accessible only for the duration of the connection. Once the connection is interrupted, the workspace is stopped and the selected IDE is no longer available to use. The pricing of this solution will therefore primarily depend on the chosen provider for running the backend, as the Gateway itself can be downloaded for free, and on the JetBrains subscription

registered to the user, given that the IDEs by JetBrains can only be accessed when in possession of a valid license.

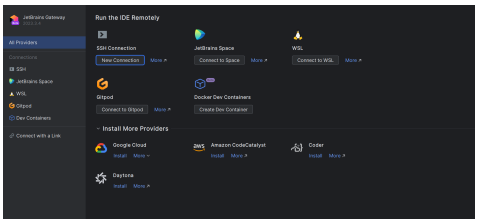

Figure 13: List of available providers on JetBrains Gateway. The above image was taken from my own instance of the gateway.

JetBrains Gateway presents a highly appealing option for those who to those who already rely on the IDEs by JetBrains for their coding tasks, as it provides all of the features already available in the locally installed versions of the JetBrains IDEs.
With respect to working with locally installable IDEs by JetBrains, it comes with the advantages of not having to store any source code on one or more machines as it is all stored in the cloud, and perhaps even more importantly, it alleviates concerns about resource limitations, since all of the processing happens on remote servers.
Amazon Web Services (AWS), a traditional cloud computing platform by Amazon, has also made its own cloud-based IDE available, under the name of Cloud9 (see Figure 14), which is aimed at writing, running and debugging serverless applications [3]. For companies and individuals that already make use of AWS as their cloud provider, Cloud9 represents a very convenient solution, as it does not imply additional charges as a service.
Only charges for compute and storage are applied, which can consist of, for instance, the charges related to the Amazon Elastic Compute Cloud (EC2) instance (which is essentially a virtual machine) used to run and store the code [3]. In short, the only applicable charges are the regular rates applied to the AWS resources that are created and/or used within the Cloud9 development environment [3].

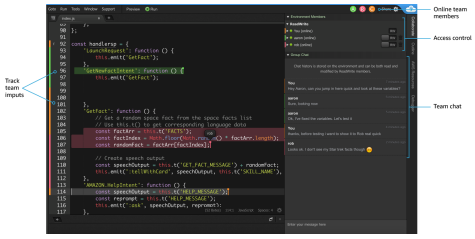

Figure 14: Cloud9's user interface and live chat [3].

In conclusion, cloud IDEs are one among the many branches of cloud computing, which is defined as the on-demand availability of computing resources on the Internet.
Cloud IDEs represent a pivotal advancement in the realm of software development, offering a great amount of advantages, which include the elimination of local infrastructure requirements, accessibility from any machine with an Internet connection, collaborative editing features, and on-demand scalability, or elasticity.
Such positive aspects can significantly optimize resource utilization and minimize costs, which is a massive selling point for companies.

2.3 Arguments Against the Use of Cloud IDEs

While it is undeniable that employing a cloud IDE offers a myriad of benefits for both individual developers and corporations, it is also important to consider the potential risks and challenges associated with the adoption of cloud computing technologies instead of traditional on-premises solutions.
The most evident disadvantage cloud IDEs have against their locally installed counterparts is their dependency on a stable Internet connection, which is a factor that often cannot be guaranteed.
Security and privacy are also primary concerns for most companies and people. Since data is stored on remote servers, there is a constant threat of data breaches and access to sensitive information (such as passwords or API keys) by unauthorized parties. Moreover, the access to the source code could compromise the developers' intellectual property [86].
For those reasons, it is also important to recognize that, while cloud IDEs come with the key advantage of not having to store the various projects' source files in a single machine and not having to rely on that machine's hardware to run resource-intensive tasks, the fact remains that storing source code in the cloud demands a high level of trust in the selected development environment provider [8]. It also requires that developers making use of cloud IDEs take additional precautionary measures, in order to avoid falling victim of malicious attacks.
Cloud adoption in general, especially in enterprise settings, can amplify the vulnerability to various types of security threats. The main threats individuals and principally companies are subject to are:

- Data breaches, in which unauthorized users manage to obtain access to sensitive information, either through stolen credentials, malware infections, social engineering, or vulnerabilities of different kinds [53].
  Some of the key threats leading to data breaches are compromised credentials and cloud accounts with admin privileges, erroneous configurations of cloud resources, malware infections, lacking activity monitoring, supply chain compromises (which include vulnerabilities in add-ons such as third-party libraries and dependencies incorporated within the cloud environment) and flaws in network configurations [53];

- Data loss, which consists of instances where data is either corrupted, eliminated or made inaccessible by malicious users. Data loss can have widely different causes, including human errors, disasters, hardware failures, ransomware (malware attacks with the aim of receiving a ransom in exchange for the mitigation of the situation) and improper management of the data lifecycle (for instance, when data is retained for durations exceeding its originally intended storage period) [53];

- Malicious insiders, where users with authorized internal access intentionally misuse their own privileges to perform malicious actions [53];

- Denial of Service (DoS) attacks, that aim to make cloud resources unavailable by overloading the infrastructure with unnecessary traffic, thereby preventing the normal functioning of the services hosted in the cloud [53];

- Account hijacking, where cloud login credentials of authorized users are stolen through techniques such as phishing, password dumps or malware, and are subsequently exploited in order to gain unauthorized access to cloud accounts, data and resources [53];

- Advanced Persistent Threats (APT), which consist of highly sophisticated targeted attacks where attackers will enter the cloud computing environments of target companies and keep secretly doing so for long periods of time, in order to steal data and/or intellectual property, as well as to observe the company's activities [53].
  Several entry points can be exploited to perform APT attacks, such as penetration through networking layers, phishing e-mails, or even physical intrusions [53];

- Attacks due to insecure interfaces and APIs, having gaps such as inadequate authentication and authorization [53];

- Attacks due to weak identity and access management, which enable security issues like compromised credentials and broken authentication [53].

When it comes to cloud IDEs in particular, there are two specific kinds of attacks that both individuals and companies working specifically with them should be well aware of, those being: [58].

1. Remote access attacks, which can also be performed through phishing techniques, where access to the Remote Desktop Protocol (RDP) of the user's system will be granted to the attacker.
   Remote access attacks (excluding those initiated through phishing, which can happen for instance when it is the user that puts him or herself at risk for clicking on malicious links) can be prevented by the user simply by not granting remote access to their system, and enhancing security by enabling Multi-factor Authentication (MFA) to their cloud IDE instances [58].

2. Browser hijacking attacks, in which the attacker will steal user data present in the browser and spy on the user over time.
   Once a browser is hijacked, every keystroke can potentially be transmitted to its attackers. Session IDs may be compromised, and sensitive information (which includes plaintext secrets) will be visible to the attacker on the user's screen. The user can protect him or herself from browser hijacking attacks by installing browser extensions coming exclusively from trusted sources, and always keeping their browsers up-to-date in order to get the most recent security patches [58].

Although they are valid concerns, cybersecurity matters are not the only security and privacy-related threats one may face when employing a cloud IDE, either. Though extremely convenient, the fact that a cloud IDE can be accessed from any machine anywhere in the world greatly increases the risk for physical security breaches as well, which may even surpass the risk for threats of digital nature [58].
Working from machines which are not our own (such as a friend's laptop, or the computer in a hotel) or that are potentially insecure, can lead to the access of source code by unauthorized subjects [58]. This is is why it is vital to keep in mind that, although cloud IDEs come with the benefit of being accessible from any machine, that does not mean that they should be accessed from just any machine at all.
Vendor lock-in is also a point of concern for companies opting for cloud IDEs, because they limit these companies to the customization and options made available by the provider of the development workspace [8]. The costs of transitioning to a different cloud IDE escalate as the companies grow, potentially anchoring them to the initially chosen one.

For those reasons, transitioning to a cloud-based solution may have an impact on flexibility, which is why it is a decision that necessitates careful consideration in terms of long-term strategic implications.
It is also important to note that, although cloud IDEs may appear to be a more cost-effective option with respect to locally installed alternatives, especially due to the absence of hardware requirements to run projects in the cloud, the expenses of adopting a cloud-based solution may still be substantial, depending on the usage requirements of each company.
In summary, while cloud IDEs offer a myriad of benefits, it is crucial to acknowledge and address the inherent potential issues that come with making use of cloud computing technologies.
Drawbacks include potential concerns regarding security and privacy, the constant reliance on a stable Internet connection, customization options limited to what is offered by the specific provider, vendor lock-in and additional costs.
Recognizing both the advantages and drawbacks of cloud computing empowers individuals and companies to effectively leverage these technologies for their own benefit.

3 The Future of Software Development: The Integration of Artificial Intelligence within Integrated Development Environments

So far, we have delved into IDEs and their main features and characteristics, as well as their migration to the cloud. Next, we will explore the integration of AI-powered tools within them, the current limitations of these tools, and how they can contribute to an enhancement of developer workflows, and as a consequence, to an increase in organizational efficiency. We will also be taking a look at future projections in the realm of artificial intelligence, keeping our focus on the relationship between AI and software development.

3.1 Artificial Intelligence and its Current Relationship with IDEs

The integration of AI-powered features within the software development workflow represents a significant shift in how developers tackle coding challenges [11].
Examples of features powered by AI currently integrated within several popular IDEs are the following:

- Context-awareness, meaning that, by performing an analysis of the codebase as a whole, in-IDE AI-powered tools can now also understand the specific context in which they are operating [11];
- Intelligent code generation, which consists of generating simple to complex snippets of code with minimal input from users [11];
- Guidance when it comes to code refactoring (restructuring existing code) [11];
- Predictive debugging, which consists of analysing the code as it is being written and anticipating potential issues and bugs before the code is compiled [11];

- Natural user interfaces, such as the previously mentioned in-IDE AI Assistant by Replit under the name of Replit AI, through which developers can interact with their IDEs using human language and receive responses in a conversational way [11].

It is also important to note that a fundamental characteristic of AI-driven tools is their ability to learn from and adapt to a changing environment. These tools consistently assess developer behavior, code modifications, and ongoing trends in the software development industry to refine their suggestions and recommendations as time goes by [11].
This continuous adaptation allows the tools to evolve and improve in conjunction with developers, maintaining their relevance and efficacy in spite of evolving requirements and coding standards [11].
The implementation of AI-powered tools and features within modern IDEs can be referred to as AI-driven development, and the development environments in which that implementation occurs can be labeled as AI-Driven Development Environments (AIDEs) [31].
A prime example of this trend is, as previously mentioned, the integration of AI-powered coding assistants within IDEs, the so-called in-IDE assistants. A few popular instances of such integration are the forementioned Replit AI by Replit, GitHub Copilot (see Figure 15) by GitHub, and JetBrains' AI Assistant, which all act as a sort of co-developer able to answer queries and assist with actions concerning the specific project at hand.
Replit AI can be used for free within the Replit cloud IDE, while GitHub Copilot is available through a subscription model as an extension that can be installed within certain IDEs and code editors.

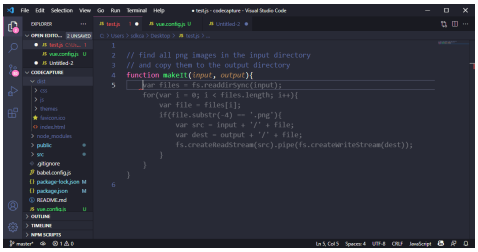

Figure 15: Code suggestions by GitHub Copilot used within Visual Studio Code [22].

JetBrains currently offers AI integrations under their AI Service Licensing, which consists of a paid service that connects the users of JetBrains products to different Large Language Models (LLMs). Under this service, the JetBrains team has recently launched their in-IDE AI Assistant, which can be integrated within any one among the various IDEs built by JetBrains, in exchange for a monthly fee (the subscription fee to JetBrains AI Service) [44].
JetBrains' AI Assistant (see Figure 16) is possibly one of the most complete and powerful in-IDE assistants available at the moment, containing features which include the generation of code snippets according to user instructions, the composition of commit messages, the explanation of pieces of code, the signaling of errors within the codebase and suggestions for code refactoring [44].
To perform these tasks, the AI Assistant leverages different LLMs (most of which made by JetBrains itself), with OpenAI serving as its primary external provider. According to JetBrains, an Enterprise plan will also soon be available, which is expected to offer even more advanced custom features [44].

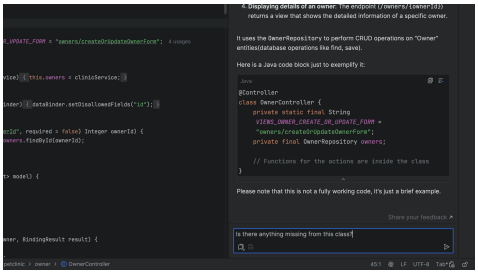

Figure 16: JetBrains' AI Assistant [44].

Among the features shared by the three tools are real-time suggestions and multi-file context awareness (which entails the ability to give suggestions based on the entire specific codebase) [38]. Replit AI also includes a collaborative AI chat, through with more teams can collaborate on the same project while also making use of the AI-powered tool [68].
In-IDE AI-powered Assistants such as the forementioned tools leverage the capabilities of Large Language Models (LLMs), which are foundation models (meaning models trained on amounts of data large enough to make them applicable across a wide range of domains) trained on massive amounts of data, which enables them to understand and generate natural language, as well as other types of content [40].
LLMs are a subset of a broader field under the name of Natural Language Processing (NLP), which consists of a combination of computational linguistics (a discipline specializing in the computational modelling of natural language) and machine learning techniques, thereby allowing computers to comprehend and generate speech and text [41].
They are now widely recognized for the role they played in popularizing generative AI, as well as driving organizations to embrace it across various applications and business sectors [40], such as education, healthcare, art, music and of course technology [39]. ChatGPT (see Figure 17), an AI-driven chatbot built by OpenAI (a research organization specializing in the field of AI) stands at the forefront of this movement.

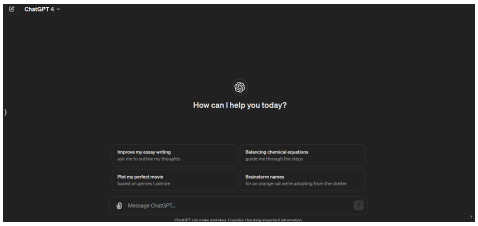

Figure 17: ChatGPT's interface, with GPT-4 as the selected model. The above image was taken from my own instance of ChatGPT.

ChatGPT marks a major breakthrough in generative AI, being the inaugural instance of an AI-driven tool drawing widespread attention in public media. It also marks the very first time in which the general public has embraced an AI-powered technology on such a broad scale.
The journey of ChatGPT started all the way back in 2018, when the very GPT-1 (with GPT standing for Generative Pre-Trained Transformer, the framework for developing generative AI), the first model behind the chatbot, was introduced in June of that year, consisting of 117 million parameters (which refer to the adjustable weights that the model uses to make its predictions). The model used unsupervised learning in the realm of language understanding

tasks, and used books as the training data to anticipate the subsequent word in a given sentence [57].
In February 2019, GPT-2 was released, representing a massive upgrade with respect to its predecessor, having a total of 1.5 billion parameters. GPT-2 was eventually launched to the public in November of that same year [57].
In June 2020, GPT-3 was released. Having been trained on 175 billion parameters, it represented a very significant advancement in comparison with the previous two models. It could be used in a very wide array of applications and settings, and was also able to translate between different languages and answer factual questions [57].
The most recently released model, GPT-4, carries on ChatGPT's tradition of ongoing enhancement. Although it is at the moment available exclusively to users subscribed to a paid plan on ChatGPT, it includes advanced features such as connectivity to the Internet, which enables real-time research capabilities [57], as well as image generation using the DALL-E tool (OpenAI's neural network specifically designed to generate images).
GPT-4o, where *o* stands for *omni*, which means *all*, was announced on May 13 2024 as OpenAI's "new flagship model that can reason across audio, vision, and text in real time" [63]. Limited access to the model is currently also offered to users in the free tier.
Unlike GPT-4, GPT-4o is able to understand any kind of input (image, text or audio) and is also capable of generating any kind of output [63]. It is therefore able to recognize objects in real life and respond accordingly, and engage with users through vocal interaction in an incredibly natural and human-like speed and tone of voice.
Users subscribed to the premium plan on ChatGPT also gain full access to the GPT Store (see Figure 18), which is a sort of AI app store where the users of ChatGPT can share their own instances of ChatGPT aimed at performing specific tasks and powered by GPT-4 as their underlying model.

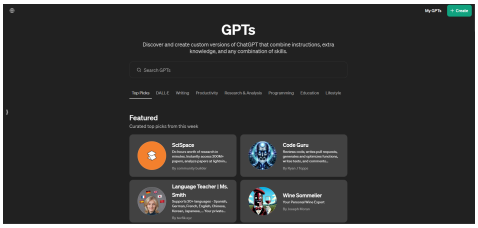

Figure 18: *Top Picks* section within ChatGPT's GPT Store, where it is possible to find featured and trending GPTs belonging to different categories, made either by the users of ChatGPT, or by the ChatGPT team itself. The above image was taken from my own instance of ChatGPT.

The emergence of AI in software development has inaugurated a new era of collaborative development, fostering previously unseen opportunities of shared value creation between AI systems and humans [39].
Cooperation between humans and AI is essential for harnessing the benefits of generative AI tools. For example, within educational environments, educators can utilize ChatGPT for scientific instruction, ensuring rigorous assessment of AI-generated materials before their incorporation into the students' curricula. In contexts where problem-solving skills are involved, generative AI can aid in brainstorming and refining solutions. Instead, within the healthcare sector, ChatGPT exhibits potential in providing support and assistance when it comes to medical practices, as well as providing insights in public health settings [39].

The integration of AI within user interactions where AI is not a mere tool but a fully fleshed collaborator can be referred to as Human-AI Experience (HAX), a discipline that entails a significant transformation in the way in which people engage with computer systems [74].
HAX highlights the increasing significance of comprehending and crafting interactions that are progressively more powered by AI, possibly initiating a new era within the realm of Human-Computer Interaction (which refers to an area of study centered on enhancing the usability of computer systems by studying the interaction between humans and computers) [74].
In this new era, the role of AI is not limited to coding assistance only, but it also encompasses the provision of intelligent insights and recommendations to the developer [74], who is now able to interact with the AI tool not only to leverage it for executing predetermined or routine tasks, but also with the expectation that it can actively contribute to the creative and problem-solving dimensions of software development tasks [39].
This sparks an interest not only in enhancing the functional aspects of developers' experiences but also in comprehending and improving the dynamics of collaboration between developers and AI inside the IDE [74].
For instance, by delegating distinct responsibilities to AI assistants, such as designating one as a Python expert, another one as a backend engineer, and a third one as a frontend engineer, software development teams can establish an effective hybrid intelligence [39].
This arrangement entails both AI and humans concentrating on different aspects of the software engineering process, which fosters cohesive teamwork. This kind of approach increases efficiency and stimulates innovative solutions within the realm of software development [39].
Of course, that does not necessarily mean that programmers will automatically adopt the suggestions given and the code written by AI assistants. Liang et al. state that only about 20 to 30% of developers will actually accept GitHub Copilot's suggestions when it comes to Python, JavaScript and TypeScript [56].
Reasons for that vary from instance to instance, but among them are the concern that the generated code may contain errors, may be hard to understand, may be difficult to edit and debug as prior knowledge of the coding principles used may be absent, or may not comply with the style of coding employed within the specific project [56].
In their explorative qualitative study aimed at understanding the practices undertaken by programmers when using AI-powered assistants to write code, Liang et al. were able to identify six main observations when it comes to the usage of such assistants:

1. Developers making use of GitHub Copilot report, on average, about 30% of their code having been written with its help [56];

2. Developers have reported that the primary motives behind their choice to use AI assistants to code are the reduction of key-strokes, greater ease to remember syntax and reduced time to finish a specific task [56];

3. The main reasons why some developers refuse to employ AI coding assistants are that the tools fail to meet certain functional (the functionalities demanded by the end user that must necessarily be integrated within the system [33]) or non functional (the quality constraints that the system must fulfil, with varying priority depending on the project [33] requirements), and that it can be challenging to guide these tools to generate the preferred outputs [56];

4. The most common usability issues of these tools among developers are that they cannot trace back to the inputs that generated the outputs given by the tool, and that, as previously mentioned, it can be quite difficult to control the tool to come up with outputs and suggestions that are truly useful [56];

5. Developers will usually give up on actually using the generated code due to the correct actions not being performed, or the snippets not meeting functional or non-functional requirements [56];

6. Developers seek to improve their interaction with AI coding assistants through both offering feedback to refine or customize the model, as well as enabling these tools to develop a deeper comprehension of code context, APIs, and programming languages in general [56].

The participants in their study also reported situations where they most succeeded in using AI-driven coding assistants, which were the following:

- Writing repetitive code, such as boilerplate code (snippets of code that are present in several different spots with very little variation) [56];

- Writing code with banal logic, while the tools performed poorly when it came to code with more complex logic [56];

- Code autocompletion, a very common use case for AI-powered tools [56];

- Quality assurance tasks, such as the generation of log messages [56];

- Proof-of-concepts, where AI assistants were capable of generating multiple solutions to a given problem [56];

- Learning new programming languages or new libraries and frameworks, where AI tools were employed rather than video tutorials and documentation or material available online [56];

- Remembering syntax relating to languages and/or methods that developers were familiar with but encountered difficulty in recalling [56];

- Improving efficiency and productivity, given that these tools could help the software development process to run more smoothly and with fewer setbacks and interruptions [56];

- Code annotation to provide context for other developers working on the same codebase by generating documentation, which led to an improvement in the collaboration between developers working on the same project [56];

- Facilitating consistency within the project when it comes to coding style, for instance by aiding programmers to maintain a clean code style throughout the codebase, as well as being able to quickly refer to sources built within the same project [56].

The main challenges faced by the participants of the study when employing AI coding assistants can instead be attributed to a a few different factors, such as a lack of knowledge with respect to what part of the input passed by the user has lead to the output returned by the AI assistant, a tendency to give up on using the generated code due to it either not performing the intended task or not fulfilling the functional or non-functional requirements of the project, and the presence of issues when it comes to controlling the model being employed to generate the desired output [56].
Developers who took part in the study also listed the techniques that they use in order to lead AI coding assistants to output the best possible results, which consist mainly of providing the assistant with very clear explanations under the shape of comments to tell it exactly what the requested code should output [56].
They also mentioned different strategies such as following code conventions (for instance, giving each function a very unique and distinguishable name), breaking down their instructions to the assistant (by splitting their prompts and the overall logic into simpler and shorter subsets), adding code for the AI tool to complement or referencing existing code within the codebase for context, and the use of prompt engineering (the science of producing the right instructions to feed the AI tool with in order to generate the best possible results) techniques, such as modifying and tweaking the instructions given to the assistant [56].
A different study by Tan et al. has also delved into the realm of AI-driven coding assistants (referred to as ACATs, which stands for AI coding assistant tools), and was aimed at exploring effectively how far these tools are from meeting the needs of developers. A few different findings were underlined by the study, among which the following:

- AI coding assistants elevate the overall task completion rate while diminishing the average time required to conclude a task [83];
- ACATs are also able to enhance the quality of the code, with GitHub Copilot showcasing the most exceptional performance in that regard. GitHub Copilot also has a tendency of recommending longer snippets of code than other AI-powered coding tools [83];
- Most participants find that these tools alleviate perceived task difficulty, enhance self-performance, decrease the need for external assistance, and generally improve the programming experience. As a consequence of that, over half of the participants (55%) report that programming feels more challenging without the assistance of ACATs [83];
- The acceptance rate (whether the code outputted gets accepted by developers) when it comes to AI coding assistants varies significantly with the type of task involved [83];
- Developers tend to accept ACATs' recommendations to fulfil functional requirements, find development ideas and diminish keystrokes. However, they often also reject their suggestions, due to those containing irrelevant or erroneous code [83];
- According to the developers who participated in the study, *correct syntax* and *similarity to correct code* stand out as the most important criteria in the evaluation of an ACAT. That is because *correct syntax* ensures that the generated code is free of errors, while *similarity to correct code* guarantees that it complies with the logic of the program and the way it is supposed to operate [83];

- Participants in the study also encountered difficulties when making use of AI coding assistants, such as inadequate performance in handling complex logic, limited support for various types of completions, restricted comprehension of natural language and different preferences regarding the length of code recommendations. Additionally, non-functional problems such as delayed response times and user interface issues were reported, which had a negative impact on the overall user experience [83];

- Among participants' expectations when it comes to ACATs are a better understanding of natural language, debug accuracy and code suggestion ranking (being able to select the most relevant code snippets), as well as the ability to learn the coding style used within the codebase. A desire for multi-modal (which entails feeding the AI model with different types of data) capabilities, a more user-friendly design and shorter times to generate outputs was also expressed [83].

Of course, as with any novel technology, the implementation of AI within software development also raises several concerns and ethical considerations, with some of those being:

- Bias-related issues, which make up a pivotal concern when it comes to AI systems, as they can can inadvertently adopt biases from their training data, which may result in discriminatory outcomes. Developers must be ready to address bias issues by cultivating diverse and representative datasets, employing fairness-aware algorithms, and regularly auditing AI models to identify and rectify biases. Ensuring equitable and inclusive AI practices is a significant challenge that demands serious attention [62];

- The surge in popularity of Generative AI having placed significant pressure on the providers of AI tools to meet the growing demand for the this type of service. As an example, in response to this growing demand, ChatGPT has introduced both a premium service and a free service with intermittent availability. JetBrains, on the other hand, has yet to effectively tackle this issue with their AI Assistant [70].
  This is particularly disappointing for developers who rely on various JetBrains products and feel as they are missing out on a very interesting feature. Despite being seamlessly compatible with the range of JetBrains IDEs, the AI Assistant's adoption is hindered by an unpredictable waiting list, which makes its widespread usage impractical [70].

- AI technologies offering potential for both constructive and detrimental applications, which raises serious concerns regarding their potential for their two-fold use and exploitation by malicious entities. For that reason, developers must conscientiously weigh the ethical ramifications of their endeavors and contemplate the potentially harmful effects of AI-powered solutions. Therefore, proactive measures should always be taken to avoid misuse and advocate for responsible utilization of AI technologies [62];

- The performance of AI systems often times relying on the access to substantial amounts of personal data. As AI applications proliferate, apprehensions emerge surrounding the methods of data acquisition, retention, and utilization [87].
  Within the realm of AI applied to software development, but also AI in general, safeguarding individuals' privacy and rights emerges as a primary concern, which demands the implementation of robust measures against data breaches, as well as unauthorized access to sensitive data [87];

- Transparency and accountability, which are also both extremely important concepts when it comes to AI systems due to them enabling a better understanding of how decisions are made by those systems, and of who should bear the responsibility if they were to make mistakes or cause harm to anyone [87].
  This is especially significant when it comes to higher risk fields, such as, for instance, autonomous vehicles and healthcare, given that, in those kinds of applications, it can often be quite difficult to determine who should be held accountable in case of incidents [87];

- AI systems implemented within the development of tools and applications in the previously mentioned safety-critical sectors must prioritize reliability and safety at all costs. Developers should engage in thorough testing and risk assessment protocols in order to effectively pinpoint and mitigate any potential safety risks. In these kinds of settings, ensuring that AI systems function safely and reliably across diverse scenarios is paramount [62];

- The challenge of determining ownership also plays a critical role in integrating AI tools into software development, and it continues to evolve as AI progresses at a pace surpassing the ability of regulators to keep pace. It therefore becomes increasingly intricate to ascertain the actual authorship of code that is generated by AI, particularly as AI tools slowly advance to be capable of autonomously generating complete software applications [87];

- Job displacement, which is a primary concern when it comes to the employment of AI tools, since they have an increasing potential to replace jobs that are currently undertaken by human workers [87].
  However, it can also be argued that AI has the potential to welcome many more jobs than the ones it could conceivably take. Even so, dealing with the effects of job displacement requires taking proactive steps, such as implementing retraining initiatives and enacting policies to ensure a fair transition for impacted workers, which calls for very efficient support systems, both social and economic [87].

In summary, we explored how AI is currently being utilized within modern IDEs, listing key features like intelligent code generation and predictive debugging. We also discussed some of the AI-based tools that contribute to AI-Driven Development Environments (AIDEs), such as GitHub Copilot and Replit AI.
Additionally, we introduced concepts like Large Language Models (LLMs) and generative AI, which led to the development of the tool we now know as ChatGPT. We emphasized the significance of Human-AI Experience (HAX) and examined developers' current perceptions of AI-powered coding assistants, as well as how close these tools are to fully meeting developer needs.
We have also talked about the ethical implications and potential concerns associated with the employment of AI-powered tools and solutions. Refining these tools is a crucial step towards realizing their full potential and irreversibly reshaping the programming and software development landscape.

3.2 The Impact of In-IDE AI-Driven Tools on Developer Satisfaction, Productivity and Consequent Organizational Efficiency

Previously, we have discussed the current capabilities and limitations of in-IDE AI-powered tools, which, at the moment, come in the shape of coding assistants.
The main purpose of building and subsequently making use of similar tools is to achieve better results with less effort, which is a consequence of an increase in the productivity of developers [49].
A McKinsey study shows how tools based on generative AI have lead to a significant increase in the speed with which developers are able to perform some of the more common development tasks. However, the study also shows that these gains in task completion speed can vary greatly based on the complexity of the task at hand (see Figure 19). This can be due to, for instance, an absence in familiarity with a necessary technology [23].

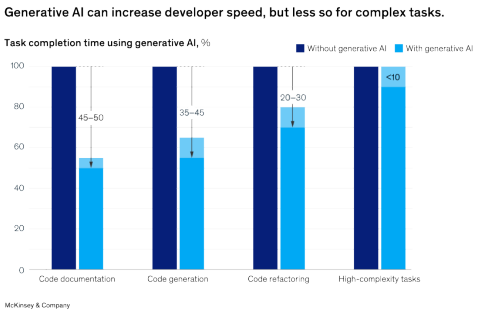

Figure 19: Task completion time using generative AI by McKinsey & Company. Developer speed is increased less consistently for more complex tasks [23].

The study also underlined that the quality of the generated code was not at all sacrificed for the improved speed in its writing. However, although the quality of the code was not compromised, participant input suggests that developers had to engage in continuous iteration with the tools to attain such quality, which suggests that the technology is most effective as a complement to developers rather than a substitute to them [23].
Similarly to the previously discussed studies by Liang at al. and Tan et al., the McKinsey study also highlighted the situations where generative AI has demonstrated to be the most useful:

- Handling routine tasks such as autocompletion of code snippets and writing documentation in a given format, which enables developers to focus on the more complex business and software challenges [23];
- Writing the very first draft of a new block of code, which allows developers to get started quicker [23];
- Helping developers get familiar with unknown codebases, frameworks and languages significantly faster, explaining novel concepts and summarizing new information, which enhances their ability to tackle new coding challenges (see Figure 20) [23];
- Facilitating modifications and improvements to already existing code [23].

Besides the more intuitive productivity enhancements, McKinsey's research also demonstrated how the use of generative AI significantly improves the coding experience of developers (see Figure 21), which is both a cause for and a consequence of the increased productivity [23].

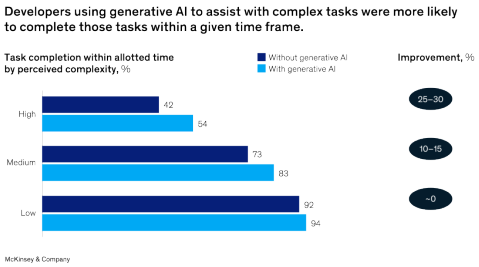

Figure 20: Developers using generative AI–based tools to perform complex tasks were between 25 and 30% more likely than those not using these tools to perform the same tasks within the given amount of time [23].

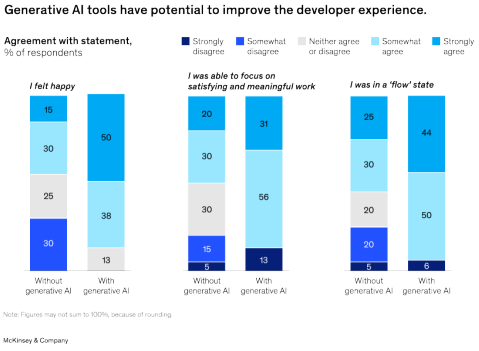

Figure 21: User responses to survey investigating how generative AI tools can improve the overall experience for developers who regularly utilize them. [23].

The study also underlined the tasks for which developer expertise is still fundamental:

- Code examination for bugs and other errors, since, sometimes, the suggestions coming from the generative AI-based tools have even brought new errors into the codebase. The developers who took part in the study mentioned how they had to guide the tools throughout the debugging process [23];
- Navigating difficult requirements, since the AI tools are much more suitable for dealing with simple coding tasks, such as optimizing code snippets, rather than more complex ones, like combining different frameworks. Participants in the study underlined how generative AI is most unhelpful when the larger organizational context needs to be taken into consideration [23];
- Considering the organizational context, given that the tools are not aware of the specific context of the specific company where they are being employed [23].
  Organizational context is especially important when it comes to making sure that the solution integrates with the remaining applications in the company and satisfies its

requirements of various types, according to the company's target users. It is, at the moment, completely up to the developers to manually provide this sort of context to the tool [23].

In conclusion, the McKinsey team recommends that developers employing generative AI-driven tools undergo a mixture of training and coaching in order to make the best possible use of these tools [23].
Training best practices include the proficiency in prompt engineering as well as the acknowledgement of risks involved with the use of generative AI-powered tools. Developers who have less than a year of coding experience should also go through additional coursework in order to be able to achieve the level of productivity of senior developers, by whom they should receive ongoing coaching [23].
Team meetings are also extremely important in order to incentivize continuous learning among developers, as well as to make sure that best practices are shared among all members in the specific organization, which contributes to a better overall quality of the generated prompts [23].
The study also enphasized how the impact of generative AI can be shared across several different coding tasks that go well beyond code generation, such as, for instance, code refactoring tasks [23].
As the productivity of developers is increased with the use of AI-powered tools, these developers will have to be scaled to tasks of higher value. Examining the current productivity levels of the company and consistently tracking its progress can unveil emerging capacities within the business [23].
Business owners will need to consider how to use that additional capacity and what sort of training is necessary to close the skill gaps that may arise [23].
McKinsey's study also underlines the need for careful risk management when it comes to the use of generative AI-based technologies within enterprise settings [23].
Among the risks that undergo a significant increase when generative AI technologies are being employed are data privacy issues (for instance, in the case that users expose personal or confidential information in their communication with the tools), behavioural vulnerabilities in the AI tool (which could be caused, for instance, by the insertion of malicious code in the public domain, in order to have an impact on the training of LLMs), as well as security vulnerabilities potentially contained in the code generated by AI. As it was previously stated, ownership debates also constitute a potential concern when it comes to the reliance on AI-generated material [23].
Another similar study was conducted by the team at GitHub, under the shape of a survey with the purpose of assessing the effects of employing GitHub Copilot. The study was conducted considering three main points:

1. Looking at productivity using a broad perspective, taking into consideration the many different factors that are able to influence it [49];

2. Including the individual perspective of developers through multiple rounds of research that included both quantitative and qualitative data [49];

3. Evaluating the effects of the usage of Copilot in day-to-day development settings by designing tests that revolved around the most typical tasks professional developers go through in their average day [49].

At the end of their study, the GitHub team came with the following insights (see Figure 22):

- Productivity goes beyond only speed, and heavily depends on both the satisfaction of developers and the conservation of mental energy. Participants in GitHub's survey in fact reported feeling significantly more fulfilled with their own careers and less frustrated while coding, which allowed them to focus on tasks that bring more satisfaction when using Copilot [49].
  They also reported that using the tool helped them save mental effort while performing coding tasks that they consider as repetitive [49].

- Although productivity is not solely dependent on speed, the study revealed that developers were able to complete tasks (especially the more repetitive ones) significantly faster while using Copilot [49].
  In order to observe that in practice, GitHub took a sample of 95 professional developers and split it into two groups [49].
  The experiment consisted of counting the amount of time it took both groups to build an HTTP server in JavaScript, where one group used Copilot to perform the task, while the other group could not. Both groups were already familiar with the technologies to be used, and were given the exact same set of instructions [49].
  The group that was allowed to use Copilot had a higher task completion rate (78%, against 70% in the group that could not employ the tool). The developers who used Copilot also completed the task 55% faster than those who did not [49].

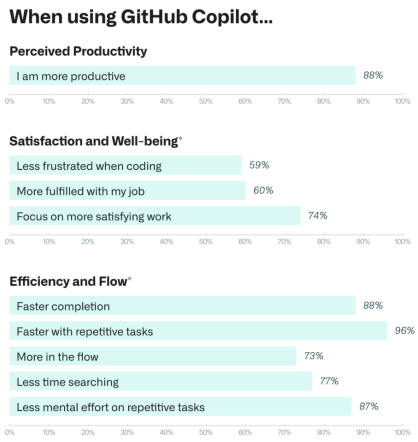

Figure 22: Responses to a survey aimed at measuring the main aspects of developer productivity when using GitHub Copilot [49].

In summary, in the study conducted by GitHub, it was observed that the usage of GitHub Copilot leads to quicker task completion times, preserves developers' cognitive energy, enables them to concentrate on tasks that are more gratifying to them, and ultimately helps them find coding tasks more fun and enjoyable [49].
Based on this analysis, it is evident that leveraging in-IDE AI tools, such as GitHub Copilot itself, yields undeniable benefits to development teams in corporate settings, as well as to developers operating individually.
Another corporate entity, an IT company specialized in digital business services, conducted a

twelve-month long experiment with GitHub Copilot in order to quantify the effects it would have on the firm's business activities [29].
Among their findings were the following, which were quite similar to the ones encountered in the two previously mentioned studies [29]:

- A boost in personal productivity by turning the less enjoyable parts of coding into more pleasant tasks [29];
- A noticeable improvement in the style and overall quality of code [29];
- A reduction in the moments of frustration, and an increase in motivation and creativity [29];
- Through the automation of the more mechanical and repetitive tasks, Copilot enabled developers to dedicate extra time to dive into more creative and especially higher-level problem solving tasks [29];
- Copilot also automatically wrote code comments and documentation for functions (see Figure 23), which facilitated the entire process of explaining modifications in the code [29];
- Copilot also did a great job in helping one of Emergn's employees with the organization and presentation of tabular data and dashboard tools. It also managed to provide structure and guidance when navigating unknown Python libraries [29];
- Exploring unfamiliar frameworks or libraries became much more manageable with Copilot, as it was able to help with the understanding of their syntax and of the various intricacies of unfamiliar structures. This saved the time that is usually wasted on minor details, facilitating a smoother learning process [29];
- Copilot was capable of adapting to the workflow of the tasks performed by each developer, as well as their individual thought processes, demonstrating an impressive ability to predict intentions [29];
- The code completion functionality sped up repetitive tasks and adjusted to the standards established by past code completion tools. In contrast to conventional code completion, it demonstrated to be capable of understanding the style of an entire codebase, which greatly simplified the task of maintaining consistency when multiple individuals are collaborating on the same codebase [29];
- Copilot managed to adjust to several different development scenarios and technologies, ranging from frontend development with React, to backend development with GraphQL, Node and NoSQL [29];
- The recently introduced chat interface, which closely resembles conversational AI tools such as ChatGPT, offered a more interactive and user-friendly environment. This could assist developers in automating various tasks, including organizing documentation, and addressing and debugging snippets of code [29];

- Especially for junior developers, Copilot could potentially become a very valuable guide, providing suggestions and generating code for unfamiliar tasks. This speeds up coding tasks and fosters a hands-on learning atmosphere tailored to real-world scenarios [29].

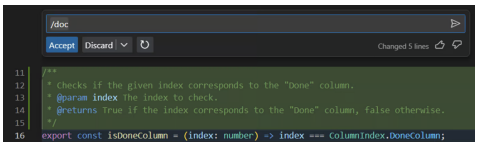

Figure 23: GitHub Copilot generating documentation for a function [29].

Emergn's experiment therefore illustrates how incorporating AI tools into everyday coding practices boosts efficiency and enhances the developer experience. Copilot's predictive, suggestive, and streamlining capabilities serve as evidence of AI's capacity to amplify productivity and innovation in software development [29].
In conclusion, we have delved into the potential of AI-driven tools to improve both the level of happiness and productivity of developers on an organizational level.
We have discussed the results of three different studies in that regard, and their conclusions were quite similar.
In general, AI has been demonstrated to have a positive impact on the productivity and overall satisfaction of developers, and as a consequence, on the companies where they are employed. However, AI is most helpful when it comes to simpler, more repetitive, and less creative tasks, which can include, for instance, generating code suggestions, optimizing snippets of code, and drafting documentation related to a specific function.
Instead, for more complex and effort-intensive tasks, such as adapting to a specific business context and navigating complicated requirements, human input and supervision are still always required.

3.3 Envisioning the Future Role of Artificial Intelligence in the Realm of Software Development

Formerly, we have discussed what the applications of artificial intelligence within modern IDEs currently looks like. However, it is also imperative to extend our analysis to encompass the potential future projections concerning the application of AI-powered tools within the domain of software development.
The application of AI-driven tools in that domain are likely to drastically change the way in which teams write, debug and deliver software. Specifically, it can be envisioned that:

- Developers will have the option of focusing on how platforms operate in goal-oriented design instead of focusing solely on the outcome of the written code, meaning that they will be able to consider the broader platform or ecosystem within which their software operates [84];
- New programming languages are likely to prioritize features tailored to facilitating a smoother integration with AI. In that case, we could therefore witness the emergence of languages featuring integrated functionalities and libraries tailored for machine learning models, neural network structures, or data processing frameworks specialized in AI-related tasks [85];

- At some point, it might be possible to build entire applications through mere natural language commands [85];
- AI tools will help with the more basic aspects of UI design, while leaving the more complex tasks in interactive design to human workers [84];
- By being able to generate high volumes of code, AI will contribute to Continuous Delivery (CD) through the increase in the overall delivery rate of developers [84].
  AI will therefore enable true Agile (a software development methodology that emphasizes the need for continuous improvement and delivery) by allowing development teams to deliver their work following a continuous flow [37];
- With larger amounts of code being produced, testing will become a higher priority [84];
- Future AI-based coding tools could be fed by a particular company, thereby generating intelligence that is focused on that enterprise's specific needs and characteristics [20];
- Custom AI-powered coding assistants could be built with the purpose of diminishing the learning curve for new platforms and tools and act as specialized chatbots [20], which could eventually also be extended to new programming languages and frameworks;
- AI tools could eventually become fundamental collaborators when it comes to the detection and subsequent mitigation of bias within the codebase that may include stereotypes or favoring a social group over another, which both contribute to an increase in social inequality [85].

As we had the chance to see in the last section, at the moment, AI's growing role within the realm of programming and software development predominantly lies on the emergence of AI-powered coding assistants.
However, it is reasonable to envision that in the near future, AI coding assistants will progress to the point of functioning as fully-fleshed development AI *agents* (a term used to describe completely autonomous AI-based systems capable of performing complex tasks with little to no human intervention) capable of managing an entire software development project entirely on their own.
An example of an AI tool that is currently moving in that direction is Devin AI, created by Cognition Labs, a software development startup company specialized in the development of AI technologies. Devin is currently advertised as "the world's first fully autonomous AI software engineer" [2].
It is expected to be capable of managing an entire software development project, from ideation to completion [2], and to perform tasks including, but not limited to:

- Building and deploying applications from start to finish, adding any new features requested by the interested party as the application development progresses [93];
- Learning how to use technologies that are not familiar to it [93];
- Spotting and subsequently fixing bugs within large codebases [93];

- Addressing feature requests and bugs within GitHub open source repositories with only a link to the specific issue, as well as contributing to mature production repositories [93];
- Training and fine-tuning its own underlying AI models given only a link to a GitHub repository [93];
- Taking up actual software engineering gigs on freelancing platforms such as Upwork [93].

All of the above mentioned capabilities have been demonstrated on short videos that can be found on Cognition Labs' website.

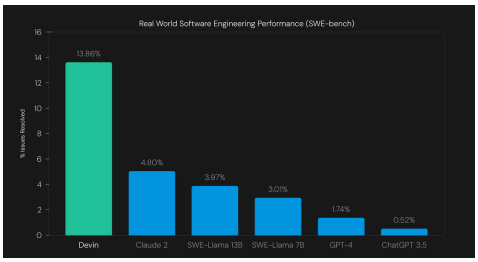

Figure 24: Devin's performance was evaluated on SWE-bench, a benchmark for AI solving real-world issues on GitHub open-source projects, where Devin managed to successfully and thoroughly resolve 13.86% of the issues [93].

Devin is also said to be able to plan and subsequently execute engineering tasks requiring several decisions, also being capable of recalling context and learning from its mistakes like any human engineer would [93].
It is also capable of engaging in dynamic collaboration with users by offering real-time progress updates, accepting user input and cooperating with the user in navigating design decisions as required [93].
Its creators have also provided Devin with the most common development tools, such as a code editor and a shell, all contained within a sandboxed environment [93].
GitHub is working on a similar initiative under the name of GitHub Workspace, which is currently defined as a development environment where the entire software development process can be executed in natural language, thereby contributing to significantly lowering the entry barrier for who can write software. The Workspace will also be accessible from mobile devices, and everything it can propose is expected to be fully editable by the user [26].

Previously, we have mentioned the possibility that, in the foreseeable future, AI coding assistants could evolve to become AI agents able to manage complex software projects entirely on their own.
The key distinction between the generative AI tools we currently use, like ChatGPT, and AI agents, lies in their operational mechanism. While these tools primarily respond to prompts inputed by humans, agents are engineered to think and work autonomously: all humans are supposed to provide them with is a specific purpose for them to pursue in their actions. Essentially, agents will prompt themselves and constantly evolve and adapt in order to fulfil the given goal in the best possible way [66].

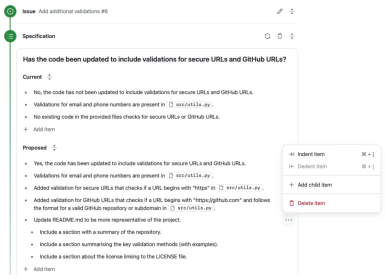

Figure 25: Plan proposed by Copilot Workspace to solve an issue concerning a potential update aimed at including validations for secure and GitHub URLs [26].

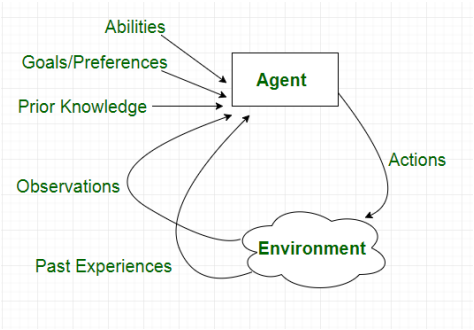

Figure 26: Characteristics of an AI agent [32].

In that sense, it can be said that AI agents are an important step towards Artificial General Intelligence (AGI) [66], which is at the moment difficult to imagine, but can be defined as a type of AI tool that is able to reason, learn, behave and generally perform the same tasks that humans do [82].
Examples of applications of AI that can be considered agents are, for instance:

- Self-driving cars, given that they are capable of bringing passengers from point A to point B, all while attaining to traffic rules and dealing with unexpected situations in traffic [66];
- Personal assistants, designed to be capable of helping humans with various (usually repetitive and routine) tasks, such as sending e-mails and scheduling meetings [32];
- Autonomous robots, which are physical agents built to help humans with more hands-on tasks, such as cleaning, organizing and delivering goods [32]. Tesla's AI-driven humanoid robot, Optimus or Tesla Bot, is an excellent example in that regard. The bot has been shown performing several kinds of tasks on video, such as carrying packages, watering plants and even folding clothes.

Agents can be split into five distinct classes based on their level of recognized intelligence and skill:

- Simple Reflex agents, which act exclusively based on their current perception of the world, meaning that they are not able to access the history concerning what they have

perceived up to date. Simple Reflex agents therefore have very limited cognitive capabilities [32];

- Model-Based Reflex agents, which operate through the identification of a condition-matching rule for the present scenario. They create a model or representation of the environment to anticipate the consequences of their actions [32];
- Goal-Based agents, which base their decision on how close they are from a specific goal, or desirable situation. Their actions are therefore tailored to minimizing their distance from this goal [32];
- Utility-Based agents, who make their decisions on which action to take depending on the utility, or preference, for each attained state. In this context, utility represents the level of "happiness" achieved by the agent [32];
- Learning agents, which are agents capable of learning from their past mistakes and experiences. Learning agents have four main components, which are the *learning element* (the component responsible for making improvements in the agent's performance, by learning from the outside environment), the *critic* (which describes how well the agent is actually doing, with respect to a performance benchmark), the *performance element* (a component responsible for selecting the external action that ought to be performed) and the *problem generator* (which is responsible for determining which actions will lead to useful experiences) [32].

At present, AI-powered tools integrated within IDEs can write code snippets based on provided instructions, and are quickly progressing towards the ability to generate entire software projects autonomously.
In the coming years, it is expected that AI agents will be capable of working together and collaborating as a team in multi-agent environments or systems, which are systems made out of multiple agents collaborating to reach a common purpose [32].
Agents in a multi-agent system would be able to, for instance, run an entire organization with significantly reduced human supervision and participation. Apart from software development, multi-agent systems could potentially be used in a myriad of different applications, such as transportation and infrastructure building, which could lead to great improvements in overall efficiency, as well as a reduction in costs [32].
They could be either homogenous (where all of the agents have the same purposes and capabilities) or heterogenous (where agents can have different purposes and capabilities), cooperative (working together in order to reach a common purpose), competitive (operating against each other in order to achieve their own purposes in the most favorable manner) or even a mixture of both cooperative and competitive behavior (a context where agents must learn to balance and combine their own interests with the interests of the larger community of agents) [32].
Within a multi-agent system, agents could also be organized in a hierarchy, where higher-level agents control and oversee the behavior of lower-level agents by providing them with specific constraints and goals, while they carry out the tasks for those purposes [32].
Multi-agent AI-based systems could therefore truly revolutionize the way society currently operates. By leveraging the collective intelligence of teams of AI agents working together, we could unlock unprecedented levels of efficiency, innovation, and adaptability across various sectors in our society.

In conclusion, we have explored the exciting landscape of future projections when it comes to the role of AI within IDEs, and within software development in general.
In that regard, we can envision that, in the near future, it might be considered normal to write an entire software program by relying solely on natural language commands, which is something that is becoming increasingly feasible with the emergence of tools like Copilot Workspace.
At the same time, new programming languages will most likely contain and prioritize features that enable a smooth integration with AI-driven tools, and that AI-powered tools themselves might become fundamental collaborators when it comes to identifying issues related to bias and ethics within the codebase where they operate.
Additionally, with the arrival of tools like Devin, the first AI-based software engineer, it is very likely that the AI tools that we currently know as coding assistants will eventually all become AI agents specialized in software development, which are autonomous AI-driven systems able to complete programming tasks without human intervention, and even collaborate together in teams by constituting multi-agent systems.

4 Envisioning the Future of Software Development Careers in the Era of Artificial Intelligence

In the previous chapter, we have discussed artificial intelligence and its role within modern IDEs. We have examined how AI is currently being integrated into these tools, its impact on developer satisfaction and efficiency, as well as future projections when it comes to its integration in the field of software development.
In this concluding chapter, we will explore the potential for the implementation of AI within the realm of software development to act as a complete replacement for software developers, the degree to which human supervision and intervention could remain essential in software development projects, and finally, the opportunities that AI-driven tools present for both software developers and non-technical individuals.

4.1 Do AI-Powered Tools Mark the End of Software Development as a Career?

In the previous chapter, we have explored how in-IDE AI-powered assistants like GitHub Copilot enhance developers' daily workflows. By handling the more mundane aspects of coding, these tools improve efficiency as well as job satisfaction, allowing developers to focus on more complex and creative tasks.
We have also examined how AI-based tools are evolving into fully autonomous agents, designed to perform specific tasks. For instance, Devin AI, an AI-powered software engineer, has demonstrated to hold the ability to independently take on job postings on freelancing platforms like Upwork.
Given the evolution of these tools towards greater autonomy and independence, to the point of being capable of managing an entire project by themselves, people currently involved in the IT industry cannot help but wonder whether the need for human intervention and supervision even in software projects of higher complexity will soon become completely obsolete.
As of now, it is reasonable to conclude that, while AI can operate independently, human

supervision is still necessary to a certain extent. In fact, it is still possible to find several job openings on LinkedIn (see Figure 27) for software development positions, even in locations like San Francisco, where it is safe to assume that the use of AI technologies is more widespread than in most places in the world.

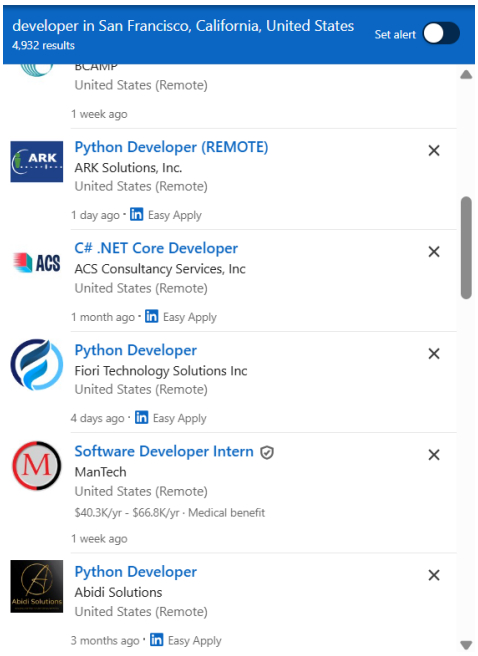

Figure 27: List of job openings for developer roles in San Francisco, CA on LinkedIn. The above image was taken upon my own visit to LinkedIn.

Figure 27 suggests that, at least for the moment, companies are either not employing AI technologies or, if they are, they are using AI to complement and enhance the capabilities of their human workers instead of to completely replace them.
IMF has conducted an interesting analysis (see Figure 28), examining the potential impact of AI technologies on the global job market, distinguishing between advanced economies, emerging economies and low-income countries [36].
Although their research does not specifically concern software developers, it provides valuable insights into how countries at various stages of economic development will be affected by AI [36].
Globally, about 40% of the job market is exposed to AI to some degree. Although in the past automation and technology have maintained a tendency to affect mostly manual and low-skilled labor, what distinguishes AI from past technological advancements is its ability to have an impact on high-skilled professions [36].
In fact, at the moment, it is capable of performing tasks that are mostly performed in the realm of white collar jobs, such as image and code generation.
IMF also emphasizes how, for that reason, it is natural that the more advanced economies will be able to benefit the most from AI technologies [36], as they are where most of the opportunities for high-skilled workers are concentrated.
In advanced economies, around 60 % of jobs may be affected by AI. Approximately half of these jobs could benefit from AI integration, and in this case AI would serve as a productivity enhancement tool [36].

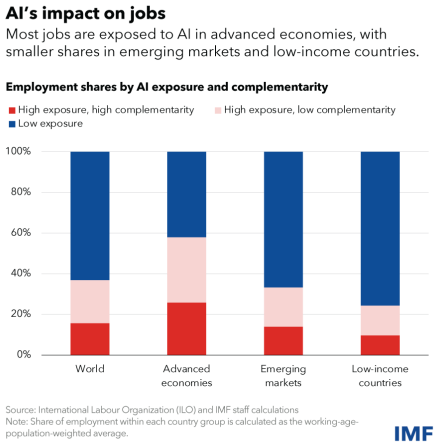

Figure 28: The impact of AI on the job market: shares of employment by exposure to AI and complementarity to it. Image credit: IMF [36].

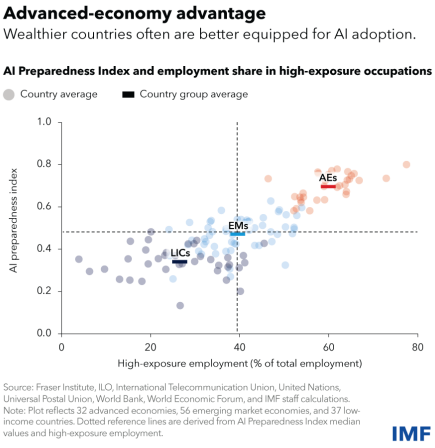

Figure 29: AI Preparedness Index measuring preparation for AI, and employment share in high-exposure occupations. Image credit: IMF [36].

However, for the remaining half, AI may take over tasks that are currently performed by human workers, potentially reducing labor demand, leading to a decrease in wages, as well as fewer job opportunities. In extreme cases, some jobs might even disappear entirely [36].
In contrast, the exposure to AI technologies in emerging markets and low-income countries is expected to be 40% and 26%, respectively. Such findings suggest that emerging markets and developing economies may face fewer immediate disruptions due to AI [36].
However, many of these countries lack the infrastructure and skilled workforce to fully benefit from AI, which could eventually lead to an increase in the inequality among nations [36].
In that regard, IMF's researchers have also developed an *AI Preparedness Index* (see Figure 29) that measures how ready each country is for the adoption of AI, in several different areas. Their findings revealed that most of the wealthier economies, in particular Denmark, Singapore and the United States, are significantly better prepared for the adoption of AI than those of low-income countries [36].
When it comes to the threat that AI tools pose to job security specifically in the software development field, a study by Kuhail et al. aimed at assessing ChatGPT's effects on software development and the perception of programmers when it comes to AI tools found that, on a scale of 1 to 5 (with 1 meaning *no threat not at all* and 5 meaning *high threat*), the average *current* job security threat among participants in the study stood at 1.89 (see Figure 30) [52]. In particular, 47.5% of participants in the study do not currently view AI as a threat to their job security, and only 3% of them currently see AI as a high threat in that regard [52].
However, the average *future* job security threat was significantly higher at 2.41 (see Figure 30), and 4% of participants viewing AI as a high threat to their job security in the future [52].

Participants to which AI does not represent a threat to job security listed some of the reasons leading to that belief, among which:

- Domain knowledge being required for development [52];
- AI tools performing well exclusively when it comes to small and simple tasks, with

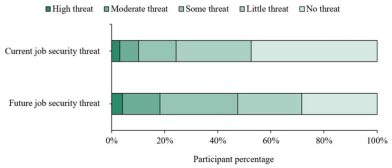

Figure 30: Participants' perceived current and future job security threats [52].

human intervention being essential for more complex and lengthy tasks [52];

- Understanding the specific context of given requirements is a task that requires a human mind [52];
- AI tools relying on knowledge and experience coming from humans in order to submit useful queries [52].

Instead, participants to which AI does represent a threat to job security listed the following reasons why they embrace that conviction:

- Given that these tools are already reasonably good at programming, there is no reason why they could not become capable of writing entire programs in the near future [52];
- Even if they might not be capable of writing an entire program on their own, AI-powered tools are likely to still be able to replace entry-level software development jobs [52];
- Given their current ability to understand human instructions, are also likely to become capable of processing custom requirements independently [52].

In summary, the study has revealed that over two-thirds of participants do not perceive an immediate job security threat coming from AI tools. However, when considering the situation in the future, about half of the participants anticipated a moderate to high job security threat, which highlights a significant difference between current and future perceptions of job security threats [52].
Those concerned about job security threats pointed out that AI tools are now capable of writing code and could potentially replace entry-level software development jobs. Conversely, participants who did not feel threatened believe that AI tools cannot replace human programmers due to the necessity of strong domain knowledge and human imagination and originality for effective software development to take place [52].
On a related note, participants in the study had different opinions concerning the acquisition of coding skills in light of this new AI era [52].
While some participants feel as there is no need to change the way in which programming is taught as AI should only be used as a last resort, others pointed out how the traditional methods used to teach the discipline are no longer the most appropriate [52].
According to the latter group, foundational topics should be taught in conjunction with AI tools, with a focus on developing problem-solving skills and understanding the limitations of these tools [52].
A different study by Vaillant et al., aimed at qualifying the perceptions of developers concerning the impact of ChatGPT in their field, also delved into their personal concerns when it comes to AI's potential to replace them in their roles [88].

In that regard, 43% of participants in the study believe that ChatGPT and related technologies may lead to developer layoffs in the future, with 11% of participants being certain of that (see Figure 31).

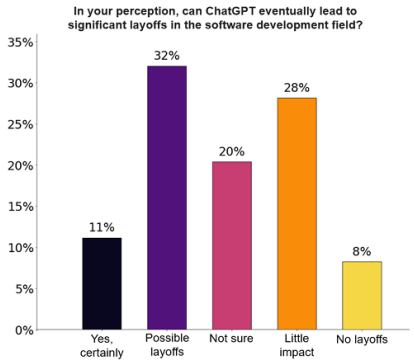

Figure 31: Perceived impact of ChatGPT on layoffs in software development [88].

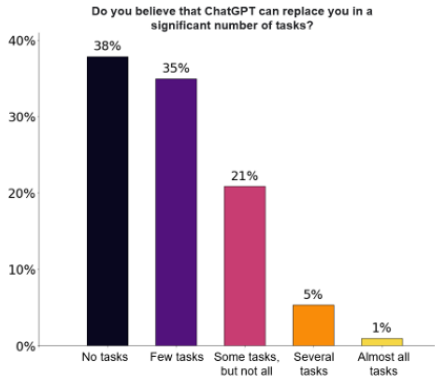

Figure 32: Perceived potential replacement by ChatGPT in software development tasks [52].

Instead, when it comes to their perceptions concerning how much the demand for software developers would be affected by AI, 23% of participants stated that they thought it would undergo a decrease, with 3% of them envisioning a significant decrease.
It is also worth noting that 73% of participants demonstrated to believe that they can either not be replaced by AI-powered tools like ChatGPT in any task whatsoever, or only in a limited amount of tasks (see Figure 32) [88].
Some participants expressed their concerns regarding the situation, calling for a necessity for stricter regulation in the field of AI. Others pointed out how, in the future, a single developer might be able to take on the workload of several developers, thereby greatly cutting costs and, as a consequence, rendering large development teams completely unnecessary within most companies [88].
The reduced need for larger teams is very likely to have a negative impact on the demand for professionals in the software development field, which will contribute to a limitation in the job opportunities in software development [88].
The study concluded itself with a list of implications for both researchers and software developers.
According to the authors, developers should:

- Anticipate a shift in their tasks towards the supervision and integration of AI technologies in software development processes [88].
  Developers should therefore embrace the trends in automation by understanding that, while certain parts of their workflow may be automated, their expertise is likely to remain largely useful [88];

- Balance coding and reviewing by staying up-to-date when it comes to the latest trends in AI-assisted software development and staying informed about potential shifts in em-

ployment dynamics, while also investing in tools to best spot and correct errors made by AI technologies [88];

- Regularly take part in discussions revolving around the regulation of AI technologies by, for instance, proposing guidelines that ensure the safe use of AI technologies combined with the respect of intellectual property and maintenance of job stability[88];
- Integrate AI-based tools such as ChatGPT in their personal programming workflows, while also staying vigilant when reviewing code generated by such tools, and avoiding to rely on them for activities deemed as security-sensitive [88].

Instead, researchers in the field of AI are encouraged to:

- Explore ways of improving the performance of AI tools when it comes to undertaking complex programming tasks, given that, so far, tools like ChatGPT have demonstrated to perform best with simple tasks [88];
- Optimize the equilibrium between programming efficiency and quality assurance in order to avoid negative situations, such as errors and hallucinatory behaviors (which consist of the generation of misleading information by AI as if it were a fact) [88];
- Contribute to the development of frameworks and guidelines aimed at protecting intellectual property and guaranteeing the safety of the users of AI tools [88];
- Mitigate the limitations to the performance of AI tools given by the absorption of dated information and data [88].

In a research paper aimed at showcasing the potential as well as the risks and limitations of AI-based tools [72], Sauvola et al. have instead identified four different scenarios when it comes to the adoption of AI technologies in different software development operations:

1. Humans will oversee all roles, utilizing tools and development environments to facilitate automation. They will be responsible for managing processes, as well as designing, implementing, testing, delivering, and maintaining products. AI tools will instead assist in automating tasks ranging from code discovery to deployment [72];
2. Although humans will still manage to dominate AI, it will be able to manage increasingly larger and more complex work areas. AI tools will be employed to automate selected parts of manual and repetitive tasks, such as code generation, documentation, testing, and deployment. Additionally, AI will be used to assist humans in tasks like design, troubleshooting, and decision-making [72];
3. AI will begin to take on specific roles, such as managing processes, as well as tasks involving design, implementation, testing, delivery, and maintenance. Meanwhile, humans will be able to focus exclusively on the most complex tasks and oversee the entire operation, ensuring it functions correctly and produces the desired results [72];
4. AI will be capable of handling various roles in software development operations, automating most or all tasks present in the software development life cycle. Humans will instead oversee and control the process, focusing mostly on operational management, problem-solving, quality assurance, and security. AI will automate most of the other tasks present in the software development lifecycle [72].

Savoula et al. employed these four scenarios to assess the potential adoption of AI in the following cases:

1. *Legacy maintenance and renewal of existing systems* falls under Scenario 1 because these systems often require manual upkeep and involve outdated APIs, architectures, and even programming languages [72].
   While human experts will continue to play critical roles, GPT and code interpreter capabilities will be able to assist developers in understanding older code within systems [72].
   They will manage to aid in debugging, testing, and analysis during maintenance operations and even in creating new adapter modules for system renewal. This will increase the likelihood that Scenario 2 will eventually prevail [72];

2. *Starting from scratch with products devoid of legacy systems* is particularly suitable for Scenarios 2 through 4, because of the absence of outdated architectures and languages. AI tools will be fully utilized for a wide range of tasks, such as code generation, debugging, testing, deployment, and maintenance [72].
   In larger software operations, Scenario 2 is more likely to unfold in the short to midterm, as AI increasingly automates repetitive tasks and even customer-facing activities [72].
   When it comes to simpler applications, there are numerous offerings with no-code environments or code generation services. In the long term, Scenario 3 and later Scenario 4 will become more probable as AI tools and their integration into processes mature [72];

3. *Networked applications and services* demand low latency, and high responsiveness and reliability. Scenario 2 is the most suitable in this situation, considering AI as a design tool to strategically plan and optimize complexities and APIs [72].
   Transitioning to Scenario 3 later on can offer developers numerous benefits, such as dynamically adapting to diverse configurations, creating customer and language variants, optimizing on-site performance, and conducting updates [72].
   In this case, achieving Scenario 4 is likely to involve leveraging emerging computing paradigms like network-level edge computing and spatial computing environments. Human roles will primarily revolve around design, architecture, and ensuring integrity, security, and performance [72].
   While many non-functional requirements or other issues can be anticipated, human operators are still needed to intervene during unprecedented challenges [72];

4. *Special software development operations* will probably initially follow Scenario 1, being a domain that will require highly skilled engineers and developers due to high service-level agreements, non-functional requirements, interoperability challenges, and real-time system interdependencies [72].
   As software assets are transitioned to repositories, platforms, and development environments, a shift to Scenario 2 can be anticipated, inheriting features from Scenario 3 [72].
   Subsequently, AI will be capable of assuming roles that alleviate human burdens by handling operational tasks, configurations, updates, and recovery, thereby consolidating Scenario 4. In that case, AI will be capable of analyzing and rectifying the behavior of complex as well as interconnected systems with real-time performance [72].

According to Sauvola et al., generative AI holds significant potential across various levels of software development operations, serving as a strategic tool that can be used by businesses and individuals with the purpose of enhancing productivity, optimizing existing resources, as well as successfully achieving significant savings when it comes to both cost and time [72]. Since the introduction tools like ChatGPT, the software development community has adopted these technologies at an impressive rate. Nevertheless, further research still needs to be conducted in order to explore the practical implementation of these proposed scenarios (Scenarios 1 through 4) within different types of software development operations in the IT industry [72].
A different research paper by Ciniselli et al. takes on a more predictive approach, forecasting how AI will transform the routine tasks of software developers by 2030 [12].
For that purpose, the authors envision an entity called *HyperAssistant*, defined as an augmented AI-driven assistant that is capable of offering consistent and thorough support to developers in 2030 by addressing current issues and limitations such as mental health issues, detection of errors, interaction between team members, development of new skills and code optimization [12].
HyperAssistant is envisioned to be capable of assisting software developers in the following five areas, leading to a significant increase in their level of productivity:

1. **Improvements in developer mental health:**

    - At present, the mental health of developers is often overlooked and not taken seriously enough. *HyperAssistant* would be able to monitor the activity levels and cognitive performance of developers in real-time, which would allow it to identify signs of stress and fatigue and intervene accordingly by suggesting a break [12];
    - It would also be capable of improving IDEs through the personalization of visual elements, in order to reduce eye tiredness and enhance the readability of such elements [12].

    By giving equal importance to mental well-being and technical skills, developers would be able to cultivate a more sustainable approach to software development, resulting in increased creativity, productivity, and job satisfaction in the long run [12].

2. **Enhanced fault detection mechanisms:**

    - *HyperAssistant* would be capable of analyzing an entire codebase in order to search for security vulnerabilities and to recognize patterns indicative of security faults [12];
    - It would also be capable of automatically executing newly created functions within its own sandboxed environment in order to test their functionality [12];
    - Finally, *HyperAssistant* would assist with the automatic generation of test cases (which are instructions on how to test a system) by leveraging code coverage analysis and behavior forecasting [12].
      By scrutinizing the codebase and potential pathways of execution, *Hyperassistant* would be able to produce test inputs geared towards optimizing code coverage. This proactive method of test case generation holds the potential to enhance software quality by detecting bugs at an early stage in the development process [12].

3. **Code optimization:**

    - *HyperAssistant* is expected to be able to prevent code duplication by checking whether repeated code is present within the same codebase [12];
    - It is also expected to be capable of ensuring the consistency of code comments with the code, thereby avoiding the presence of obsolete code comments [12].

4. **Contribution to smarter team interactions:**

    - *HyperAssistant* would enhance team interactions by fostering more robust and smoother communication among developers, as well as improving task allocation. It would hold the ability to recommend that a developer seek assistance from a team member in writing a particular function, recognizing similarities in code already written [12];
    - Integrated seamlessly within the team, HyperAssistant would monitor the activities of all members and facilitate intelligent interaction among them [12];
    - Additionally, it would be capable of estimating the time needed for a specific task based on extensive observation of programming activities, thereby optimizing the coding pipeline as a whole [12];
    - It wold also play a pivotal role within the team, generating the initial project draft. Beginning with an abstract design, it would autonomously write the project code and then allocate each section to the most suitable developer based on their expertise, ensuring correctness [12];
    - Moreover, it would serve as a task reminder for developers, monitoring their progress and prompting actions such as committing changes on GitHub [12].

5. **Contribution to a smoother acquisition and development of new skills:**

    - *HyperAssistant* would be able to assist developers in staying current with available technologies by suggesting relevant articles based on the code they are working on, and eventually propose new features inherent to the task they are currently performing [12];
    - It would also be capable of recommending learning courses or tutorials to help developers address specific skill gaps, as well as generating customized tests to assess their progress [12].

Overall, the work done by Ciniselli et al. evidences the transformative impact of AI-driven tools when it comes to developer workflows by the year of 2030. According to the researchers, developers are expected to accomplish their tasks with increased efficiency, collaborate with each other with more ease and make self-care a top priority, which is inevitably bound to increase productivity and fulfillment [12].
Very much like in previously analyzed studies, Ciniselli et al. underline that, in their view, AI is much more likely to become an assistant and ally for developers than to act as a replacement for them [12].
Essentially, according to the authors, by 2030, thanks to AI-powered assistants like *Hyperassistant*, the amount of work that currently requires a full day at the office is likely to become achievable in just half a day or even less [12].

Therefore, in a similar scenario, developers will have significantly more free time, less work-induced stress, and more consciousness about the importance of adequate self-care.
The integration of AI-powered tools within the software development workflow will serve as an amplifier for the capabilities of software developers, eventually leading to the creation of software tools and solutions that are much more sophisticated, secure and reliable than what is achievable today [12].
In conclusion, while it is understandable for software developers to feel concerned about AI technologies impacting their job security, according to the existing literature on the topic, it is generally improbable that these technologies will reach a level of independence that allows them to completely replace human developers.
In fact, according to the research papers discussed, most software developers do not fear complete replacement by AI, and most researchers in the field do not at the moment predict a complete replacement of software developers by AI technologies capable of building software.
While AI tools are expected to take over several tasks currently performed by human programmers, their role is a lot more likely to be supportive rather than substitute.
This means that the involvement of human programmers is likely to endure across various fields of software development, while AI will act more of a pain reliever and productivity enhancer, rather than a replacement.
Although AI-based tools are for the moment unlikely to take over the place of programmers, it is worth noting that one of the inevitable side effects of the availability of such advanced tools, especially in-IDE ones like GitHub Copilot and Replit AI, is that the entry barrier for software development is significantly lowered.
This means that, while in the pre-AI era coding was seen as a difficult skill and landing a decent programming job was seen as simply unachievable without a solid background, we can now expect to see a lot more people looking to join the workforce as software developers.
This aspect, even more than the question of whether AI might render human-driven software development obsolete, ought to be considered both by current professionals in software development, and by those aspiring to enter the field.
As more people are capable of getting into software development thanks to the help of AI-driven tools, an increase in the already fierce competition for development jobs is unavoidable.
Overall, a spike in the supply for developers, accompanied by a likely reduction in the corresponding demand (although AI is unlikely to completely wipe out software developers, it is bound to provoke a decrease in the request for junior developers looking for entry-level positions) is a probable scenario, at least in the coming years.
At the same time, entrepreneurship has become more achievable and attractive than ever. AI-powered tools create a fertile ground for aspiring entrepreneurs to start their own businesses. As these new ventures grow, they will eventually create new job opportunities for employment-seeking individuals affected by the reduced demand for entry-level software development roles.
In that regard, Fortune Business Insights predicts that the global Generative AI market will approach nearly one trillion USD by 2032, with the majority of this expansion centered in North America (see Figure 33).
Fortune Business Insights also mention how the exponential necessity to build a virtual world in the metaverse offers a significant contribution to the advancement of the generative AI market [43].

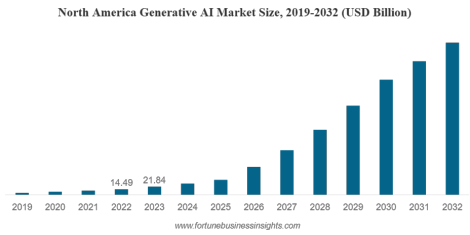

Figure 33: Projected Generative AI market size in North America, spanning from 2019 to 2032 with estimates from 2024 onward [43].

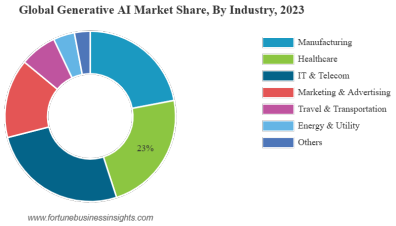

Figure 34: Global Generative AI Market share by industry in 2023, with IT & Telecom, Manufactoring and Healthcare being the leading sectors in that regard [43].

In fact, generative AI allows developers in the field of Virtual Reality (VR) to to develop an extensive collection of captivating and unique game environments [43].
It is also worth noting that the COVID-19 pandemic, in spite of its many negative impacts in several different contexts, had a significantly positive effect when it comes to the adoption of AI-driven tools and technologies [43].
As more companies shifted to work-from-home models and entire industries became completely or almost completely digital, many of them started a gradual adoption of AI technologies during the pandemic in order to increase internal productivity [43].

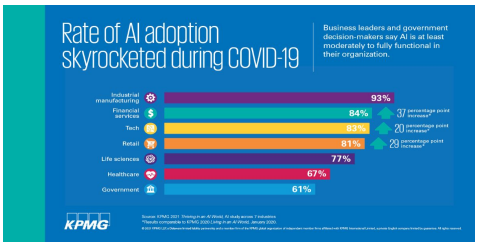

Figure 35: Increase of AI adoption following the start of the COVID-19 pandemic. Image credit: KPMG [89].

Naturally, it is safe to consider that most of the current predictions when it comes to AI-powered tools and technologies refer to the coming years and decades at most. Given the speed of technological advancement we are currently experiencing, it is close to impossible to know what not only AI, but technology in general, will look like hundreds of years from now.

4.2 Exploring the Impact of AI-Driven Tools on the Software Development Job Market from a Different Perspective

At the end of the previous section, we began exploring a different perspective on the impact of AI-powered tools when it comes to the software development job market, emphasizing how these tools lower the entry barrier for software development careers and increase entrepreneurial opportunities.
Rather than considering the potential general displacement of software developers by AI technologies, we should view the effects of these technologies on the job market more as an

indirect outcome of their implementation.
An important part of such outcome consists of a much lower entry barrier when it comes to the field of software development, given that AI tools are capable of providing those less experienced in coding with knowledge close to or equivalent to their senior counterparts' [50]. At the moment, that phenomenon is a much more realistic concern than the idea of AI replacing all software developers.
In an article about this topic, Faris Zacina argues that, while AI still requires human oversight and is unlikely to displace developers in the near future, it will significantly lower the entry barrier for founders to build new companies. This, in turn, will provoke an expansion in the available market for software developers to work in [94].
As a result, numerous new, smaller companies will emerge globally. While larger companies are more likely to downsize their development teams, developers who are laid off or unable to secure positions in these larger contexts may find other opportunities within these new smaller businesses supported by AI. In the author's own words, "In 2024, programmers won't just survive — they'll thrive" [94].
According to Zacina and many others, AI is therefore, at least for the foreseeable future, more of an opportunity than a threat for both aspiring founders and employment-seeking developers.
Y Combinator, one of the world's most well-recognized startup accelerators, enjoys a list over 200 AI-related companies funded only in the last year-and-a-half (see Figure 36).
This is a very significant figure, considering that Y Combinator funds a very limited number of startups per semester.

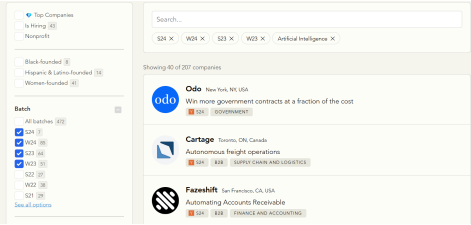

Figure 36: Number of AI-related startups funded by Y Combinator between 2023 and 2024. The service offering for these startups ranges all the way from AI-powered e-mail automation to YouTube content and movie generation. The above image was taken upon my own visit to Y Combinator's website.

An article published on website of the MIT Management Sloan School concerning a podcast involving Ethan Mollick and Bill Aulet [59] on how AI is going to change entrepreneurship emphasizes the fact that entrepreneurs face a myriad of daily tasks, from administrative work to product development. AI-driven tools are therefore especially useful in the realm of entrepreneurship because they can handle these tasks efficiently, allowing entrepreneurs to focus on their core skills [10].
Additionally, these tools can provide guidance for founders or potential founders who are unsure of their next steps in a business venture, assisting with routine tasks such as drafting emails and product development [10].
Beyond undertaking specific tasks, there is also a fundamental shift in processes, which allows for easier, quicker, and cheaper experimentation and testing. This rethinking of entrepreneurship also entails significantly shorter development timelines [10].
This new wave of AI-powered entrepreneurship will potentially render certain companies

who fail to adapt to it obsolete, and compel others to adapt their own strategies to the new landscape. Moreover, the widespread availability of AI tools ensures that global competitors are all on an equal footing [10].
The free accessibility to tools like ChatGPT builds a landscape where proficiency in English and coding is widespread, which leads to a dispersion of talent across borders and to the intensification of global competition in entrepreneurship [10].
The article also delves into Mollick's advice for entrepreneurs navigating the AI era, consisting in three main points [10]:

1. Prioritizing experimentation by testing several inputs and subsequently contrasting and comparing outputs [10];

2. Reconsidering the worth of proprietary datasets (unique data that a specific company has collected and owns exclusively), given that AI models already possess a vast knowledge base, which renders proprietary data redundant [10];

3. Remembering that entrepreneurs are suited for capitalizing on times of transformation, and that their main ability is to be capable of taking action during periods characterized by general uncertainty [10].

The creation of new companies is especially facilitated by low or no-code tools [55], such as, for instance, Langflow, the GUI for LangChain, which is a Python-based framework designed to allow developers to build applications with LLMs [54].
Langflow's creator, a startup called Logspace, has recently been acquired by DataSax, which is a company specialized in database management. The integration of Langflow and DataStax forms a comprehensive Generative AI application stack, providing flexible deployment options, including seamless integration with DataStax Astra DB, a cloud-based database as a service (DBaaS). It also offers a robust ecosystem of Python libraries [21].
The inner workings LangChain (see Figure 37) involve the concept of *chain*, which consists in a sequence of automated actions starting from the user's query and ending at the output generated by the model [6].
These chains, in turn, are made of *links*, which are the actions that are put together in order to compose an entire chain.
Langflow operates in a similar fashion, the key distinction being the utilization of a GUI in place of the reliance on hard-coded processes.
Langflow allows both technical non-technical people to build AI-powered applications manually by using a drag-and-drop type of interface (see Figure 38), involving minimal to no need for knowledge in programming.

In an article concerning the impact of AI technologies when it comes to software development, KPMG has listed some of the things that companies will now see as possibilities, also thanks to the reduction of the entry barrier for new developers on complex codebases [51]:

- Streamlining the onboarding process for large developer groups and being able to quickly integrate new developers into projects, thereby contributing to a boost in the pace of software development without having to constantly rely on existing team members for the guidance of newcomers [51];

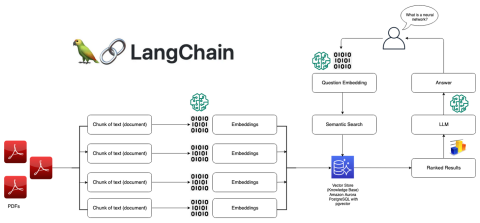

Figure 37: A graphic representation of LangChain's workflow through which a user is able to ask questions concerning various PDF documents to an LLM [6].
In this context, *embeddings* refer to vector representations of data. They essentially translate pieces of data such as images, audio and text into a mathematical form that can be given in input to machine learning models [16].

- Enhancing developer versatility across various frameworks and platforms through the support of generative AI tools, expanding the capacity of even junior developers to undertake a broader spectrum of software development projects [51];
- Fine-tuning generative AI models to incorporate the company's legacy knowledge, which diminishes the reliance on developers tied to the codebase's history and paves the way for new talents [51].

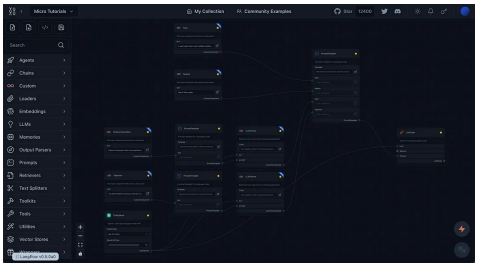

Figure 38: Langflow's user interface [61].

Because AI-powered tools are becoming increasingly capable of generating code snippets autonomously, newcomers in software development are able to leverage these tools to generate code without the necessity of fully mastering all of the complexities themselves [77].
These tools will also allow junior developers to undertake higher-level tasks, which will inevitably accelerate their learning curve and help them become more proficient, faster [50].
This entire mechanism will eventually lead to an inevitable increase in the supply for development jobs, especially those labeled as entry-level positions.
Naturally, previously mentioned, a sudden boost in the supply of junior developers in a field that is already deemed as increasingly competitive, combined with a natural reduction in demand due to AI's growing ability to effectively perform low to mid-complexity tasks, could ultimately lead to a negative impact on the tech industry employment landscape.
Apart from that aspect, the availability of tools like Devin AI, which is expected to be capable of handling an entire software project on its own, raises the question: will a basic understanding or even no understanding at all of coding soon become sufficient in order to oversee AI-driven software development projects? Or, on the contrary, will it be necessary to

hold an elevated level of expertise in order to manage projects advanced by AI?
The answer to this question is at the moment highly personal, and depends on one's beliefs about the autonomy of AI technologies. If the first hypothesis is true, that can only contribute to the thesis that, while AI-driven tools may have a negative effect on the job market, they will also make it much easier for non-technical people with innovative business ideas to become founders of tech companies.
However, although these tools could allow non-technical individuals to develop their own projects, not everyone will be motivated to invest the time needed to master them, and not everyone will be able to recognize a business need to address.
This presents a prime opportunity for those willing to overcome the learning curve to come up with interesting business ideas, and eventually build their own companies or applications.
For all the forementioned reasons, and as previously mentioned, the widespread availability of AI-powered tools presents an ideal opportunity for new startups and companies to emerge.
The expected exponential growth of the AI market and the ease of use provided by no-code and low-code tools, combined with the availability of autonomous AI tools capable of independently building software projects, allows even non-technical individuals with an innovative mindset to capitalize on this opportunity.

4.3 Final Remarks on the Future of Software Developers and Enthusiasts in an Era Driven by AI

This thesis has provided a multifaceted exploration of the integration of AI-powered tools and features within IDEs and development settings, and its implications for developers.
Starting with an overview of IDEs, we have discussed their evolution towards cloud-based solutions, highlighting the benefits and drawbacks of this transition. We have then examined the growing prevalence of AI-powered technologies in modern development tools.
In that regard, several studies conducted by researchers have revealed the potential of AI to significantly enhance developer workflows by streamlining development processes and facilitating more efficient coding practices.
However, some studies have also raised valid concerns regarding the potential risks associated with automation, including the possibility of displacing traditional developer roles, particularly in business settings.
At the same time, most of these studies have concluded that developers are not overly concerned about being entirely replaced by AI technologies. Instead, they are generally confident that their expertise will continue to be valuable in the coming years.
The last chapter has directed our attention to this precise issue, delving into whether developers could indeed be largely replaced by AI-based tools.
With regard to that, we have reached the conclusion that it is instead more worth acknowledging that these AI-driven tools will inevitably lower the entry barrier for software development. This will increase the supply of software developers, and is likely to be accompanied by a stagnation in the demand for entry-level roles.
In fact, while AI is unlikely to replace developers entirely, companies may be less inclined to hire junior developers, as their skill sets may be surpassed by what AI can accomplish.
While this may be bad news for developers, we have also highlighted that AI-powered tools create an ideal environment for new startups and companies to emerge.
This suggests that many new companies are likely to appear in the near future, creating job

opportunities for developers affected by the shift in the software development job market.
Likewise, the presence of AI-powered tools, which can potentially accomplish software projects autonomously at a human-like level, offers significant opportunities for junior developers and individuals without technical expertise. Those who are prepared to invest time in identifying a business need can consider starting their own business ventures by utilizing these tools.
We have also evidenced how, even with the availability of low-code and no-code solutions, there is still a learning curve that not everyone is willing to overcome.
Although, naturally, not everyone will manage to identify a business need to tackle, those who are willing to invest the effort to master these tools will be well-positioned to build their own startups powered by AI.
In light of this, developers should attempt to not think of AI exclusively as a looming threat destined to replace them sooner or later. Instead, they should try to view it as an opportunity.
In summary, our analysis indicates that AI-based tools will likely decrease the barriers to entry in software development, something that could could potentially lead to a decrease in the demand for software developers, especially those seeking entry-level positions. However, it also creates opportunities for new startups to emerge, consequently generating new job openings.
Despite the challenges, both technical and non-technical individuals should see AI-powered tools as a unique, once-in-a-lifetime opportunity for entrepreneurship and innovation.
Instead of adhering to traditional employment paths, enthusiasts can take a leap and leverage these tools to create new ventures and succeed in this newly unlocked era of our rapidly evolving technological landscape.

Bibliography

[1] Software Advice. *Codeanywhere*. URL: https://www.softwareadvice.com/app-development/codeanywhere-profile/.

[2] Devin AI. *Devin AI The Revolutionary AI Software Engineer by Cognition AI*. URL: https://devin-ai.dev/.

[3] AWS. *AWS Cloud9*. URL: https://aws.amazon.com/cloud9/.

[4] AWS. *What is a Public Cloud?* URL: https://aws.amazon.com/what-is/public-cloud/.

[5] AWS. *What is Cloud Native?* URL: https://aws.amazon.com/what-is/cloud-native/.

[6] AWS. *What Is LangChain?* URL: https://aws.amazon.com/what-is/langchain/.

[7] BetaWiki. *Visual Studio 97*. URL: https://betawiki.net/wiki/Visual_Studio_97.

[8] D. Bhuriya and A. Sharma. "Study on Pros, Cons and Application of Cloud Computing". In: *International Journal of Research and Analytical Reviews* Volume 6.Issue 2. URL: http://ijrar.com/upload$_issue/ijrar_issue_2$0544039$.pdf$ (2019), pp. 959–964.

[9] Erik Bledsoe. *What is a Cloud IDE?* May 28, 2021. URL: https://coder.com/blog/what-is-a-cloud-ide.

[10] Sara Brown. *How generative AI is changing entrepreneurship*. January 17, 2024. URL: https://mitsloan.mit.edu/ideas-made-to-matter/how-generative-ai-changing-entrepreneurship.

[11] S. Chinthapatla. "Unleashing the Future: A Deep Dive into AI-Enhanced Productivity for Developers". In: *International Journal of Engineering, Science Mathematics* Volume 13.Issue 3. URL: https://www.researchgate.net/profile/Saikrishna-Chinthapatla/publication/379112$Enhanced_Productivity_for_Developers/links/65fb39efa4857c7962655b43/Unleashing-the-Future-A-Deep-Dive-into-AI-Enhanced-Productivity-for-Developers.pdf$ (2024).

[12] M. Ciniselli et al. "From Today's Code to Tomorrow's Symphony: The AI Transformation of Developer's Routine by 2030". In: DOI: https://doi.org/10.48550/arXiv.2405.12731 (2024).

[13] J. Cito et al. "The Making of Cloud Applications– An Empirical Study on Software Development for the Cloud". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* Bergamo, Italy. DOI: https://doi.org/10.1145/2786805.2786826 (August 30-September 4 2015), pp. 393–403.

[14] Google Cloud. *What is Cloud Computing?* URL: https://cloud.google.com/learn/what-is-cloud-computing.

[15] Google Cloud. *What is cloud hosting?* URL: https://cloud.google.com/learn/what-is-cloud-hosting#section-2.

[16] Cloudfare. *What are embeddings in machine learning?* URL: https://www.cloudflare.com/en-gb/learning/ai/what-are-embeddings/.

[17] Codeanywhere. *Features*. URL: https://codeanywhere.com/#features.

[18] codezone. *Understanding Integrated Development Environments (IDEs) and Their Significance in Programming*. August 7, 2023. URL: https://medium.com/@codezone/understanding-integrated-development-environments-ides-and-their-significance-in-programming-244a9338b99b.

[19] Microsoft Corporation. *Microsoft Announces Visual Studio 97, A Comprehensive Suite of Microsoft Visual Development Tools*. January 28 1997. URL: https://news.microsoft.com/1997/01/28/microsoft-announces-visual-studio-97-a-comprehensive-suite-of-microsoft-visual-development-tools/.

[20] Lev Craig. *The promises and risks of AI in software development*. April 26, 2023. URL: https://www.techtarget.com/searchitoperations/feature/The-promises-and-risks-of-AI-in-software-development.

[21] datanami. *DataStax Announces Acquisition of Langflow to Transform RAG Development*. April 4, 2024. URL: https://www.cloudflare.com/en-gb/learning/ai/what-are-embeddings/.

[22] Carlos Delgado. *Testing GitHub Copilot: How good is it?* July 29, 2021. URL: https://ourcodeworld.com/articles/read/1555/testing-github-copilot-how-good-is-it.

[23] McKinsey Digital. *Unleashing developer productivity with generative AI*. June 27, 2023. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai.

[24] Digitalinteraction. *The Evolution of Integrated Development Environments: A Historical Perspective*. URL: https://digitalinteraction.co.uk/the-evolution-of-integrated-development-environments-a-historical-perspective/.

[25] Celo Docs. *Deploy on Celo with Replit*. URL: https://docs.celo.org/developer/setup/replit.

[26] Thomas Dohmke. *GitHub Copilot Workspace: Welcome to the Copilot-native developer environment*. April 29, 2024. URL: https://github.blog/2024-04-29-github-copilot-workspace/.

[27] eduNitas. *Maestro I*. URL: https://wiki.edunitas.com/IT/en/114-10/Maestro-I_20000_eduNitas.html#First_presentation_in_1975.

[28] Matt Ellis. *A Deep Dive Into JetBrains Gateway*. December 3, 2021. URL: https://blog.jetbrains.com/blog/2021/12/03/dive-into-jetbrains-gateway/.

[29] Emergn. *How AI tools impact the way we develop software: our GitHub Copilot journey*. URL: https://www.emergn.com/insights/how-ai-tools-impact-the-way-we-develop-software-our-github-copilot-journey/.

[30] encyclopedia.com. *Key To Disk*. URL: https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/key-disk#:~:text=key%20to%20disk,often%20then%20verified.

[31] N. Ernst and G. Bavota. "AI-driven Development Is Here: Should You Worry?" In: *IEEE Software* Volume 39.Issue 2. DOI: 10.1109/MS.2021.3133805 (2022), pp. 106–110.

[32] GeeksForGeeks. *Agents in Artificial Intelligence*. URL: https://www.geeksforgeeks.org/agents-artificial-intelligence/.

[33] GeeksForGeeks. *Functional vs Non Functional Requirements*. URL: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/.

[34] GeeksForGeeks. *Nano vs VIM editor – What's the difference between nano and vim editors?* URL: https://www.geeksforgeeks.org/nano-vs-vim-editor-whats-the-difference-between-nano-and-vim-editors/.

[35] GeeksforGeeks. *How to Install Notepad++ on Windows?* URL: https://www.geeksforgeeks.org/how-to-install-notepad-plus-plus-on-windows/.

[36] Kristalina Georgieva. *AI Will Transform the Global Economy. Let's Make Sure It Benefits Humanity.* January 14, 2024. URL: https://www.imf.org/en/Blogs/Articles/2024/01/14/ai-will-transform-the-global-economy-lets-make-sure-it-benefits-humanity.

[37] Sanjay Gidwani. *AI and the future of software development*. August 8, 2023. URL: https://www.infoworld.com/article/3704270/ai-and-the-future-of-software-development.html.

[38] GitHub. *GitHub Copilot*. URL: https://github.com/features/copilot.

[39] M. Hamza et al. "Human-AI Collaboration in Software Engineering: Lessons Learned from a Hands-On Workshop". In: *Proceedings of ACM Woodstock conference* URL: https://arxiv.org/abs/2312.10620.2 pages. New York, USA (June 3-5 2018).

[40] IBM. *What are large language models (LLMs)?* URL: https://www.ibm.com/topics/large-language-models.

[41] IBM. *What is natural language processing (NLP)?* URL: https://www.ibm.com/topics/natural-language-processing.

[42] Rakhim Davletkaliev Ilya Birman. *UI Museum: Turbo Pascal 7.1*. URL: https://ilyabirman.net/meanwhile/all/ui-museum-turbo-pascal-7-1/.

[43] Fortune Business Insights. *Generative AI Market Size, Share Industry Analysis, By Model (Generative Adversarial Networks or GANs and Transformer Based Models), By Industry vs Application, and Regional Forecast, 2024-2032*. (Paper Summary). May 20, 2024. URL: https://www.fortunebusinessinsights.com/generative-ai-market-107837.

[44] JetBrains. *JetBrains AI*. URL: https://www.jetbrains.com/ai/.

[45] JetBrains. *JetBrains Remote Development Gateway*. URL: https://www.jetbrains.com/remote-development/gateway/.

[46] JetBrains. *PyCharm*. URL: https://www.jetbrains.com/pycharm/.

[47] Project Jupyter. *About Project Jupyter*. URL: https://jupyter.org/about.

[48] Project Jupyter. *JupyterLab*. URL: https://jupyterlab.readthedocs.io/en/stable/.

[49] Eirini Kalliamvakou. *Research: quantifying GitHub Copilot's impact on developer productivity and happiness*. URL: https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/.

[50] Peter Klimek. *The benefits and risks of AI development tools*. September 29, 2023. URL: https://www.scmagazine.com/native/the-benefits-and-risks-of-ai-development-tools.

[51] KPMG. *The startling power generative AI is bringing to software development*. URL: https://kpmg.com/us/en/articles/2023/generative-artificial-intelligence.html.

[52] M. Kuhail et al. ""Will I be replaced?" Assessing ChatGPT's effect on software development and programmer perceptions of AI tools". In: *Science of Computer Programming* Volume 235.DOI: https://doi.org/10.1016/j.scico.2024.103111 (2024).

[53] A. Kunduru. "The Perils and Defenses of Cloud Computing: a Comprehensive Review". In: *Central Asian Journal of Mathematical Theory and Computer Sciences* Volume 4.No. 9. DOI: https://cajmtcs.centralasianstudies.org/index.php/CAJMTCS/article/view/515 (2023), pp. 29–41.

[54] LangChain. URL: https://www.langchain.com/.

[55] John Leonard. *Low code: AI is lowering the barriers to entry, report*. 5December 2023. URL: https://www.computing.co.uk/news/4152890/low-code-ai-lowering-barriers-entry-report.

[56] J. Liang, C. Yang, and B. Myers. "A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges". In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* Article No. 52. Lisbon, Portugal. DOI: https://doi.org/10.1145/3597503.3608128 (April 14-20 2024), pp. 1–13.

[57] Bernard Marr. *A Short History Of ChatGPT: How We Got To Where We Are Today*. May 19, 2023. URL: https://www.forbes.com/sites/bernardmarr/2023/05/19/a-short-history-of-chatgpt-how-we-got-to-where-we-are-today/.

[58] Dwayne McDaniel. *Securing The New Frontier in Developer Environments: Cloud IDEs*. November 21, 2022. URL: https://blog.gitguardian.com/securing-developer-environments-cloud-ides/.

[59] MIT. URL: https://player.fm/series/trust-the-process-mit/the-ai-for-entrepreneurship-guy-whartons-ethan-mollick-with-bill-aulet.

[60] Nancy Muriithi. *Infrastructure as Code - Everything You Need to Know*. January 14, 2022. URL: https://blog.gitguardian.com/infrastructure-as-code-everything-you-need-to-know/.

[61] Rodrigo Nader. *Langflow Micro Tutorials — Micro Tutorial Writer*. Sep 24, 2023. URL: https://medium.com/logspace/langflow-micro-tutorials-micro-tutorial-writer-ec4686236a63.

[62] Billy Okeyo. *The Future is Now - Exploring the Role of AI in Software Development*. February 26, 2024. URL: https://www.billyokeyo.com/posts/role-of-ai/.

[63] OpenAI. *Hello GPT-4o*. May 13, 2024. URL: https://openai.com/index/hello-gpt-4o/.

[64] Ronnie Payne. *Eclipse IDE Review*. June 30, 2023. URL: https://www.developer.com/java/eclipse-ide-review/.

[65] Jeffrey M. Perkel. *Five reasons why researchers should learn to love the command line*. 2, February 2021. URL: https://www.nature.com/articles/d41586-021-00263-0.

[66] Miguel Rebelo. *What is an AI agent?* June 1, 2023. URL: https://zapier.com/blog/ai-agent/.

[67] Replit. URL: https://replit.com/.

[68] Replit. *Turn natural language into code*. URL: https://replit.com/ai.

[69] Terminal Root. *100 Tips for the VIM Editor*. October 29, 2019. URL: https://terminalroot.com/100-tips-for-the-vim-editor/.

[70] Rowe. *How AI can and can't be used within IDEs*. URL: https://www.roweit.co.uk/ai-within-ides/.

[71] H. Sampath, A. Merrick, and A. Macvean. "Accessibility of Command Line Interface". In: *CHI Conference on Human Factors in Computing Systems* Yokohama, Japan. Article No. 489. DOI: https://doi.org/10.1145/3411764.3445544 (May 8-13 2021), pp. 1–10.

[72] J. Sauvola et al. "Future of software development with generative AI". In: *Automated Software Engineering* Volume 31.No. 26. DOI: https://doi.org/10.1007/s10515-024-00426-z (2024).

[73] M. Schröder and J. Cito. "An empirical investigation of command-line customization". In: *Empirical Software Engineering* Volume 27.No. 30. DOI: https://doi.org/10.1007/s10664-021-10036-y (2021).

[74] A. Sergeyuk, S. Titov, and M. Izadi. "In-IDE Human-AI Experience in the Era of Large Language Models; A Literature Review". In: *Association for Computing Machinery* DOI: https://doi.org/10.48550/arXiv.2401.10739 (2024).

[75] M. Sharma, T. K. Mishra, and A. Kumar. "Source code auto-completion using various deep learning models underlimited computing resources". In: *Complex Intelligent Systems* Volume 8.No. 5. DOI: https://doi.org/10.1007/s40747-022-00708-7 (2022), pp. 4357–4368.

[76] A. Shukla. "Cloud-Based Lightweight Modern Integrated Development Environments (IDEs) and their Future". In: *Journal of Artificial Intelligence Cloud Computing* DOI: doi.org/10.47363/JAICC/2024(3)218 (2024), pp. 2–3.

[77] Sonar. *a developer's guide to AI-assisted software development*. URL: https://www.sonarsource.com/learn/ai-assisted-software-development/.

[78] D. Spinellis. "Developing in the Cloud". In: *Tools of the Trade* Volume 31.Issue 2. DOI: 10.1109/MS.2014.33 (2014), pp. 41–43.

[79] Visual Studio. URL: https://visualstudio.microsoft.com/.

[80] Visual Studio. *Visual Studio Code*. URL: https://code.visualstudio.com/.

[81] Visual Studio. *Visual Studio Code Documentation*. URL: https://code.visualstudio.com/docs/editor/intellisense.

[82] Matthew Urwin Sunny Betz. *What Is Artificial General Intelligence?* January 30, 2024. URL: https://builtin.com/artificial-intelligence/artificial-general-intelligence.

[83] X. Tan et al. "How far are AI-powered programming assistants from meeting developers' needs?" In: DOI: https://doi.org/10.48550/arXiv.2404.12000 (2024).

[84] Flow Transformation Team. *AI in software development: Key opportunities + challenges*. March 4, 2024. URL: https://www.pluralsight.com/resources/blog/leadership/AI-in-software-development#replace-devs.

[85] The Upwork Team. *Will AI Replace Developers? The (Bright) Future of Software*. October 23, 2023. URL: https://www.upwork.com/resources/will-ai-replace-developers.

[86] Teamcode. *Comparing Online Code Editors and Cloud IDEs*. June 15, 2023. URL: https://medium.com/@teamcode20233/comparing-online-code-editors-and-cloud-ides-c29a0ad2ea7b.

[87] Capitol Technology University. *The Ethical Considerations of Artificial Intelligence*. May 30, 2023. URL: https://www.captechu.edu/blog/ethical-considerations-of-artificial-intelligence.

[88] T. Vaillant et al. "Developers' Perceptions on the Impact of ChatGPT in Software Development: A Survey". In: DOI: https://doi.org/10.48550/arXiv.2405.12195 (2024).

[89] VentureBeat. *KPMG: AI adoption is accelerating in the pandemic*. March 9, 2021. URL: https://venturebeat.com/ai/kpmg-ai-adoption-is-accelerating-in-the-pandemic/.

[90] S. Vinoth et al. "Application of cloud computing in banking and e-commerce and related security threats". In: *Materials Today: Proceedings* Volume 51.Part 8. DOI: https://doi.org/10.1016/j.matpr.2021.11.121 (2022), pp. 2172–2175.

[91] Google Cloud Website. *What is Cloud Native?* URL: https://cloud.google.com/learn/what-is-cloud-native.

[92] Wikipedia. *Key To Disk*. URL: https://en.wikipedia.org/wiki/Maestro.

[93] Scott Wu. *Introducing Devin, the first AI software engineer*. March 12, 2024. URL: https://www.cognition-labs.com/introducing-devin.

[94] Faris Zacina. *Forget the Hype: AI Isn't Taking Your Coding Job*. April 17, 2024. URL: https://medium.com/mop-developers/forget-the-hype-ai-isnt-taking-your-coding-job-9047f2d16171.