ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	Η χρήση της Τεχνητής Νοημοσύνης στη δημιουργία λογισμικού The Use of Artificial Intelligence in Software Development
Ονοματεπώνυμο Φοιτήτριας	Ελένη Κρητικού
Πατρώνυμο	Ελευθέριος
Αριθμός Μητρώου	Π/09064
Επιβλέπων	Παναγιώτης Τσάκωνας, Μέλος ΕΔΙΠ

Ημερομηνία Παράδοσης Σεπτέμβριος 2026

Copyright ©

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου για τη συνεχή στήριξη και τη υπομονή τους κατά τη διάρκεια των σπουδών μου.

Επίσης, ευχαριστώ τον επιβλέποντα, κ. Παναγιώτη Τσάκωνα, για την πολύτιμη καθοδήγηση, την εμπιστοσύνη και τις καίριες παρατηρήσεις του καθ' όλη τη διάρκεια εκπόνησης της παρούσας πτυχιακής εργασίας.

Περίληψη

Η Τεχνητή Νοημοσύνη αποκτά ολοένα σημαντικότερο ρόλο στη μηχανική λογισμικού, εισάγοντας νέες δυνατότητες στον τρόπο σχεδιασμού, ανάπτυξης, ελέγχου και συντήρησης των εφαρμογών. Σκοπός της παρούσας βιβλιογραφικής εργασίας είναι η διερεύνηση της χρήσης της Τεχνητής Νοημοσύνης στη δημιουργία λογισμικού, με έμφαση στις εφαρμογές, στα οφέλη, στους περιορισμούς και στις προοπτικές που προκύπτουν από την ενσωμάτωσή της στη διαδικασία ανάπτυξης. Εξετάζονται η αυτόματη παραγωγή κώδικα, η υποστήριξη των προγραμματιστών, ο εντοπισμός και η διόρθωση σφαλμάτων, καθώς και η αυτοματοποίηση επιμέρους εργασιών. Παράλληλα, αναλύεται η επίδραση των σχετικών τεχνολογιών στην παραγωγικότητα και στην ποιότητα του λογισμικού, χωρίς να παραβλέπονται ζητήματα αξιοπιστίας, ασφάλειας, πνευματικής ιδιοκτησίας, επαγγελματικής ευθύνης και περιβαλλοντικού αποτυπώματος. Ιδιαίτερη αναφορά γίνεται στους AI agents, οι οποίοι διευρύνουν τις δυνατότητες της Τεχνητής Νοημοσύνης από την απλή υποβοήθηση της συγγραφής κώδικα προς την εκτέλεση περισσότερο σύνθετων εργασιών ανάπτυξης. Η βιβλιογραφική διερεύνηση δείχνει ότι οι τεχνολογίες αυτές μπορούν να περιορίσουν τον χρόνο που απαιτείται για επαναλαμβανόμενες εργασίες και να μεταβάλουν το περιεχόμενο της εργασίας του προγραμματιστή, χωρίς ωστόσο να καθιστούν περιττή την ανθρώπινη τεχνική γνώση και κρίση. Αντίθετα, μεγαλύτερη σημασία αποκτούν η αξιολόγηση του παραγόμενου κώδικα, ο έλεγχος της ποιότητας και της ασφάλειας και η λήψη αποφάσεων σε σύνθετα προβλήματα. Συνολικά, διαπιστώνεται ότι η Τεχνητή Νοημοσύνη μετασχηματίζει τη μηχανική λογισμικού, ενώ η αποτελεσματική αξιοποίησή της προϋποθέτει συνδυασμό τεχνολογικής αυτοματοποίησης, ανθρώπινης εποπτείας και υπεύθυνων πρακτικών ανάπτυξης.

Λέξεις-κλειδιά: Τεχνητή Νοημοσύνη, ανάπτυξη λογισμικού, παραγωγή κώδικα, AI agents, μηχανική λογισμικού.

Abstract

Artificial Intelligence is playing an increasingly important role in software engineering, introducing new possibilities in the design, development, testing, and maintenance of applications. The purpose of this literature review is to investigate the use of Artificial Intelligence in software development, with an emphasis on its applications, benefits, limitations, and the prospects arising from its integration into the development process. The study examines automated code generation, developer assistance, error detection and correction, and the automation of individual tasks. At the same time, it analyses the impact of these technologies on software productivity and quality, while also considering issues related to reliability, security, intellectual property, professional responsibility, and environmental impact. Particular attention is paid to AI agents, which extend the capabilities of Artificial Intelligence beyond basic assistance with code writing towards the execution of more complex development tasks. The literature review indicates that these technologies can reduce the time required for repetitive tasks and reshape the nature of programmers' work, without rendering human technical knowledge and judgement unnecessary. On the contrary, evaluating generated code, ensuring its quality and security, and making decisions in complex situations are becoming increasingly important. Overall, the findings demonstrate that Artificial Intelligence is transforming software engineering, while its effective use requires a combination of technological automation, human oversight, and responsible development practices.

Keywords: Artificial Intelligence, software development, code generation, AI agents, software engineering.

Περιεχόμενα

Εισαγωγή

Η Τεχνητή Νοημοσύνη (Artificial Intelligence - AI) αποτελεί μία από τις σημαντικότερες τεχνολογικές εξελίξεις της σύγχρονης εποχής, καθώς η εφαρμογή της επεκτείνεται σε ολοένα και περισσότερους τομείς της οικονομικής, επιστημονικής και επαγγελματικής δραστηριότητας. Η ανάπτυξη συστημάτων που μπορούν να επεξεργάζονται μεγάλο όγκο δεδομένων, να αναγνωρίζουν πρότυπα, να παράγουν περιεχόμενο και να υποστηρίζουν τη λήψη αποφάσεων έχει μεταβάλει σημαντικά τον τρόπο με τον οποίο χρησιμοποιούνται οι υπολογιστικές τεχνολογίες. Η Τεχνητή Νοημοσύνη δεν αποτελεί μία ενιαία τεχνολογία, αλλά ένα ευρύτερο επιστημονικό πεδίο που περιλαμβάνει διαφορετικές προσεγγίσεις και τεχνικές, όπως η μηχανική μάθηση, η βαθιά μάθηση και, πιο πρόσφατα, η παραγωγική Τεχνητή Νοημοσύνη (Russell & Norvig, 2021).

Ιδιαίτερα σημαντική εξέλιξη αποτέλεσε η ανάπτυξη των μεγάλων γλωσσικών μοντέλων (Large Language Models - LLMs) και των εφαρμογών παραγωγικής Τεχνητής Νοημοσύνης. Τα συστήματα αυτά έχουν τη δυνατότητα να επεξεργάζονται φυσική γλώσσα και να δημιουργούν νέο περιεχόμενο με βάση τις οδηγίες που λαμβάνουν από τον χρήστη. Παράλληλα, η εξέλιξη μοντέλων που έχουν εκπαιδευτεί και σε προγραμματιστικό κώδικα επέτρεψε την εφαρμογή της παραγωγικής AI στον χώρο του προγραμματισμού και της μηχανικής λογισμικού. Τα μοντέλα αυτά μπορούν να δημιουργούν κώδικα από περιγραφές φυσικής γλώσσας, να συμπληρώνουν υπάρχοντα προγράμματα και να υποστηρίζουν την επίλυση προγραμματιστικών προβλημάτων (Chen et al., 2021).

Η ανάπτυξη λογισμικού αποτελεί μία σύνθετη διαδικασία, η οποία δεν περιορίζεται αποκλειστικά στη συγγραφή κώδικα. Περιλαμβάνει την ανάλυση των απαιτήσεων, τον σχεδιασμό του συστήματος, την υλοποίηση, τον έλεγχο, τη διάθεση και τη μετέπειτα συντήρησή του. Σε κάθε ένα από αυτά τα στάδια απαιτούνται διαφορετικές γνώσεις, εργαλεία και διαδικασίες, ενώ σημαντικό ρόλο έχουν η συνεργασία μεταξύ των μελών μιας ομάδας και η συνεχής προσαρμογή στις μεταβαλλόμενες απαιτήσεις ενός έργου. Η μηχανική λογισμικού επιδιώκει ακριβώς την οργανωμένη και συστηματική αντιμετώπιση αυτής της πολυπλοκότητας, με στόχο την ανάπτυξη αξιόπιστου και ποιοτικού λογισμικού (Pressman & Maxim, 2020).

Μέσα σε αυτό το περιβάλλον, η Τεχνητή Νοημοσύνη αρχίζει να αποκτά έναν ολοένα σημαντικότερο ρόλο. Τα σύγχρονα εργαλεία AI μπορούν να λειτουργούν ως βοηθοί του προγραμματιστή κατά τη διάρκεια διαφορετικών εργασιών, όπως η παραγωγή τμημάτων κώδικα, η επεξήγηση μιας υπάρχουσας υλοποίησης, η δημιουργία τεκμηρίωσης, η πρόταση διορθώσεων και η δημιουργία περιπτώσεων δοκιμών. Η εξέλιξη αυτή μεταβάλλει σταδιακά τον τρόπο με τον οποίο πραγματοποιείται η ανάπτυξη λογισμικού, καθώς μέρος της τεχνικής εργασίας μπορεί πλέον να εκτελείται με την υποστήριξη ενός συστήματος Τεχνητής Νοημοσύνης (Vaithilingam et al., 2022).

Ένα από τα βασικά χαρακτηριστικά αυτής της αλλαγής είναι η δυνατότητα αυτόματης παραγωγής κώδικα. Ο προγραμματιστής μπορεί να περιγράψει σε φυσική γλώσσα τη λειτουργία που επιθυμεί και το σύστημα να δημιουργήσει μία πιθανή υλοποίηση. Με αυτόν τον τρόπο μειώνεται σε ορισμένες περιπτώσεις ο χρόνος που απαιτείται για την παραγωγή τυποποιημένου ή επαναλαμβανόμενου κώδικα, ενώ παράλληλα παρέχεται υποστήριξη σε εργασίες που διαφορετικά θα απαιτούσαν αναζήτηση τεκμηρίωσης ή παραδειγμάτων. Η ανάπτυξη μοντέλων προσανατολισμένων στον προγραμματιστικό κώδικα έχει δείξει ότι τα μεγάλα γλωσσικά μοντέλα μπορούν να επιλύουν ένα μέρος των προγραμματιστικών εργασιών όταν τους παρέχεται κατάλληλη περιγραφή του προβλήματος (Chen et al., 2021).

Ωστόσο, η χρήση της Τεχνητής Νοημοσύνης στη δημιουργία λογισμικού δεν περιορίζεται στην παραγωγή νέου κώδικα. Μπορεί να αξιοποιηθεί για την υποστήριξη του προγραμματιστή κατά την κατανόηση ενός υπάρχοντος προγράμματος, τον εντοπισμό πιθανών σφαλμάτων, τη δημιουργία δοκιμών και την αναδιαμόρφωση τμημάτων κώδικα. Με τον τρόπο αυτό, η AI αρχίζει να ενσωματώνεται σε περισσότερα σημεία του κύκλου ανάπτυξης και όχι μόνο στη φάση της υλοποίησης. Η αλληλεπίδραση μεταξύ ανθρώπου και συστήματος αποκτά έτσι περισσότερο συνεχή χαρακτήρα, καθώς ο προγραμματιστής μπορεί να λαμβάνει προτάσεις κατά τη διάρκεια διαφορετικών εργασιών (Vaithilingam et al., 2022).

Η εξέλιξη αυτή δημιουργεί σημαντικές προσδοκίες σχετικά με την αύξηση της παραγωγικότητας των προγραμματιστών. Η δυνατότητα αυτοματοποίησης επαναλαμβανόμενων εργασιών μπορεί να επιτρέψει στους επαγγελματίες να επικεντρωθούν περισσότερο στην επίλυση σύνθετων προβλημάτων και στον σχεδιασμό των συστημάτων. Παράλληλα, η γρήγορη

δημιουργία ενός πρώτου προσχεδίου κώδικα μπορεί να επιταχύνει την έναρξη μιας εργασίας και να μειώσει τον χρόνο που απαιτείται για ορισμένες επιμέρους δραστηριότητες. Παρόλα αυτά, η αύξηση της ταχύτητας παραγωγής δεν πρέπει να ταυτίζεται αυτόματα με αύξηση της συνολικής ποιότητας ή αποτελεσματικότητας, καθώς το παραγόμενο αποτέλεσμα εξακολουθεί να χρειάζεται αξιολόγηση και προσαρμογή από τον άνθρωπο (Vaithilingam et al., 2022).

Σημαντικό ζήτημα αποτελεί επομένως η ποιότητα του κώδικα που παράγεται από συστήματα AI. Ένα μεγάλο γλωσσικό μοντέλο μπορεί να δημιουργήσει κώδικα που εμφανίζεται συντακτικά σωστός και πειστικός, χωρίς αυτό να σημαίνει ότι ανταποκρίνεται πλήρως στις απαιτήσεις του συγκεκριμένου έργου. Μπορεί να υπάρχουν λογικά σφάλματα, ακατάλληλες επιλογές υλοποίησης ή προβλήματα που γίνονται εμφανή μόνο όταν το πρόγραμμα χρησιμοποιηθεί σε διαφορετικές συνθήκες. Για τον λόγο αυτό, η ανθρώπινη αξιολόγηση παραμένει απαραίτητη και ο παραγόμενος κώδικας δεν πρέπει να αντιμετωπίζεται ως έτοιμη και εκ προοιμίου ορθή λύση (Vaithilingam et al., 2022).

Ακόμη μεγαλύτερη σημασία έχει το ζήτημα της ασφάλειας του λογισμικού. Η δυνατότητα παραγωγής κώδικα σε μικρό χρονικό διάστημα μπορεί να αποτελέσει σημαντικό πλεονέκτημα, όμως ένας κώδικας που λειτουργεί δεν είναι απαραίτητα και ασφαλής. Η έρευνα των Pearce et al. (2022) σχετικά με προτάσεις κώδικα από το GitHub Copilot έδειξε ότι σε συγκεκριμένα σενάρια μπορούσαν να παραχθούν λύσεις που περιλάμβαναν αδυναμίες ασφαλείας. Το γεγονός αυτό αναδεικνύει την ανάγκη να εφαρμόζονται οι ίδιες ή και αυστηρότερες διαδικασίες ελέγχου στον κώδικα που παράγεται με τη βοήθεια Τεχνητής Νοημοσύνης (Pearce et al., 2022).

Εκτός από τα τεχνικά ζητήματα, η ενσωμάτωση της AI στη δημιουργία λογισμικού δημιουργεί και ηθικούς, νομικούς και επαγγελματικούς προβληματισμούς. Η εκπαίδευση των μεγάλων γλωσσικών μοντέλων βασίζεται σε πολύ μεγάλες συλλογές δεδομένων, γεγονός που έχει προκαλέσει συζητήσεις σχετικά με την προέλευση και την ποιότητα του υλικού εκπαίδευσης. Παράλληλα, η χρήση δημόσια διαθέσιμου περιεχομένου δεν σημαίνει απαραίτητα ότι αυτό στερείται προστασίας από δικαιώματα πνευματικής ιδιοκτησίας. Ζητήματα που σχετίζονται με τη διαφάνεια, τα δεδομένα εκπαίδευσης και τις ευρύτερες επιπτώσεις των μεγάλων γλωσσικών μοντέλων έχουν επομένως αποκτήσει ιδιαίτερη σημασία (Bender et al., 2021).

Ένας επιπλέον προβληματισμός αφορά το περιβαλλοντικό αποτύπωμα της Τεχνητής Νοημοσύνης. Η εκπαίδευση και η λειτουργία μεγάλων μοντέλων απαιτούν σημαντικούς υπολογιστικούς πόρους, γεγονός που συνδέεται με κατανάλωση ενέργειας και ευρύτερες περιβαλλοντικές επιπτώσεις. Η προσέγγιση της Green AI υποστηρίζει ότι η αξιολόγηση ενός συστήματος Τεχνητής Νοημοσύνης δεν πρέπει να βασίζεται αποκλειστικά στην απόδοσή του, αλλά να λαμβάνει υπόψη και την υπολογιστική αποδοτικότητα και τους πόρους που απαιτούνται για την επίτευξη του αποτελέσματος (Schwartz et al., 2020).

Ιδιαίτερο ενδιαφέρον παρουσιάζει επίσης ο τρόπος με τον οποίο η Τεχνητή Νοημοσύνη μπορεί να επηρεάσει το μέλλον του επαγγέλματος του προγραμματιστή. Καθώς τα εργαλεία αποκτούν τη δυνατότητα να εκτελούν μεγαλύτερο αριθμό τεχνικών εργασιών, ορισμένες δραστηριότητες που σήμερα πραγματοποιούνται χειροκίνητα είναι πιθανό να αυτοματοποιηθούν σε μεγαλύτερο βαθμό. Αυτό δεν σημαίνει απαραίτητα πλήρη αντικατάσταση του προγραμματιστή. Αντίθετα, μπορεί να οδηγήσει σε μεταβολή των καθηκόντων του, με μεγαλύτερη έμφαση στην περιγραφή των απαιτήσεων, στην αρχιτεκτονική, στην αξιολόγηση των προτάσεων της AI και στον έλεγχο της ποιότητας του τελικού αποτελέσματος (Liu et al., 2024).

Η προοπτική αυτή ενισχύεται από την ανάπτυξη των AI agents, οι οποίοι αποτελούν μία νεότερη κατεύθυνση εφαρμογής των μεγάλων γλωσσικών μοντέλων στη μηχανική λογισμικού. Σε αντίθεση με ένα σύστημα που παράγει μία μεμονωμένη απάντηση ή ένα τμήμα κώδικα, ένας agent μπορεί να οργανώσει μία εργασία σε επιμέρους βήματα, να αλληλεπιδράσει με εργαλεία και αρχεία και να χρησιμοποιήσει τα αποτελέσματα προηγούμενων ενεργειών για να συνεχίσει την προσπάθειά του. Η εξέλιξη των LLM-based agents δημιουργεί επομένως προοπτικές για μεγαλύτερο βαθμό αυτοματοποίησης σε πραγματικές εργασίες ανάπτυξης λογισμικού (Liu et al., 2024).

Παρά τη σημαντική πρόοδο, η αντιμετώπιση πραγματικών προβλημάτων μηχανικής λογισμικού παραμένει περισσότερο απαιτητική από την παραγωγή μεμονωμένων αποσπασμάτων κώδικα. Ένα πραγματικό έργο περιλαμβάνει μεγάλο αριθμό αρχείων, εξαρτήσεις μεταξύ διαφορετικών τμημάτων του συστήματος και απαιτήσεις που πρέπει να γίνουν κατανοητές πριν πραγματοποιηθεί μία αλλαγή. Το SWE-bench, το οποίο δημιουργήθηκε για την αξιολόγηση

γλωσσικών μοντέλων σε πραγματικά ζητήματα από αποθετήρια λογισμικού, αναδεικνύει ακριβώς τη δυσκολία μετάβασης από μικρές προγραμματιστικές εργασίες στην αυτόνομη αντιμετώπιση σύνθετων προβλημάτων πραγματικών έργων (Jimenez et al., 2024).

Με βάση τα παραπάνω, σκοπός της παρούσας εργασίας είναι να εξετάσει τη χρήση της Τεχνητής Νοημοσύνης στη δημιουργία λογισμικού και να παρουσιάσει τόσο τις δυνατότητες όσο και τους περιορισμούς που προκύπτουν από την ενσωμάτωσή της στη μηχανική λογισμικού. Ειδικότερα, εξετάζεται η συμβολή της AI στην παραγωγή κώδικα, στην υποστήριξη των προγραμματιστών, στον έλεγχο και στον εντοπισμό σφαλμάτων και στην αυτοματοποίηση διαδικασιών ανάπτυξης. Παράλληλα, διερευνώνται οι επιπτώσεις στην παραγωγικότητα και στην ποιότητα του λογισμικού, καθώς και ζητήματα αξιοπιστίας, ασφάλειας, επαγγελματικής ευθύνης, πνευματικής ιδιοκτησίας και βιωσιμότητας.

Η εργασία βασίζεται σε βιβλιογραφική προσέγγιση, μέσα από τη μελέτη επιστημονικών δημοσιεύσεων, βιβλίων και σύγχρονων ερευνητικών εργασιών που σχετίζονται με την Τεχνητή Νοημοσύνη και τη μηχανική λογισμικού. Μέσα από τη σύνθεση της υπάρχουσας βιβλιογραφίας επιχειρείται να παρουσιαστεί μία συνολική εικόνα της σημερινής κατάστασης, αλλά και των τάσεων που διαμορφώνονται για τα επόμενα χρόνια. Ιδιαίτερη έμφαση δίνεται στη μετάβαση από τα απλά εργαλεία υποβοήθησης του προγραμματισμού προς περισσότερο αυτόνομα συστήματα, καθώς η συγκεκριμένη εξέλιξη είναι πιθανό να επηρεάσει σημαντικά τον τρόπο οργάνωσης της ανάπτυξης λογισμικού (Liu et al., 2024).

Ως προς τη διάρθρωση της εργασίας, στο πρώτο κεφάλαιο παρουσιάζονται οι βασικές έννοιες της Τεχνητής Νοημοσύνης, η ιστορική της εξέλιξη και οι τεχνολογίες της μηχανικής μάθησης και της παραγωγικής AI. Στο δεύτερο κεφάλαιο εξετάζεται η σύγχρονη διαδικασία δημιουργίας λογισμικού, οι τεχνολογίες που χρησιμοποιούνται και οι βασικές προκλήσεις που αντιμετωπίζει η μηχανική λογισμικού. Το τρίτο κεφάλαιο επικεντρώνεται στις πρακτικές εφαρμογές της Τεχνητής Νοημοσύνης στην ανάπτυξη λογισμικού, όπως η παραγωγή κώδικα, η υποστήριξη των προγραμματιστών, ο εντοπισμός σφαλμάτων και η αυτοματοποίηση διαδικασιών. Στο τέταρτο κεφάλαιο αναλύονται τα οφέλη αλλά και τα ζητήματα που σχετίζονται με την παραγωγικότητα, την ποιότητα, την αξιοπιστία, την ασφάλεια, την ηθική, την πνευματική ιδιοκτησία και τις περιβαλλοντικές επιπτώσεις. Τέλος, στο πέμπτο κεφάλαιο εξετάζεται το μέλλον της Τεχνητής Νοημοσύνης στη μηχανική λογισμικού, με έμφαση στην εξέλιξη των εργαλείων, στους AI agents, στον μετασχηματισμό του ρόλου του προγραμματιστή, στη συνεργασία ανθρώπου και AI και στις προϋποθέσεις για την υπεύθυνη αξιοποίησή της.

Κεφάλαιο 1: Τεχνητή Νοημοσύνη και Βασικές Έννοιες

1.1 Ορισμός της Τεχνητής Νοημοσύνης

Η Τεχνητή Νοημοσύνη (Artificial Intelligence - AI) αποτελεί έναν από τους σημαντικότερους και ταχύτερα εξελισσόμενους κλάδους της επιστήμης της Πληροφορικής. Πρόκειται για ένα διεπιστημονικό πεδίο που συνδυάζει γνώσεις από την επιστήμη των υπολογιστών, τα μαθηματικά, τη στατιστική, τη γνωσιακή επιστήμη, τη φιλοσοφία και τη μηχανική, με σκοπό τη δημιουργία συστημάτων τα οποία μπορούν να εκτελούν εργασίες που υπό φυσιολογικές συνθήκες απαιτούν ανθρώπινη νοημοσύνη (Russell & Norvig, 2016). Η σημασία της Τεχνητής Νοημοσύνης έχει αυξηθεί ιδιαίτερα κατά τις τελευταίες δεκαετίες, καθώς η πρόοδος στην υπολογιστική ισχύ, η διαθεσιμότητα μεγάλων όγκων δεδομένων και η ανάπτυξη νέων αλγορίθμων έχουν επιτρέψει τη δημιουργία συστημάτων με πρωτοφανείς δυνατότητες μάθησης, ανάλυσης και λήψης αποφάσεων (Bishop, 2006).

Παρότι ο όρος «Τεχνητή Νοημοσύνη» χρησιμοποιείται ευρέως, δεν υπάρχει ένας μοναδικός και καθολικά αποδεκτός ορισμός. Η πολυπλοκότητα της έννοιας οφείλεται στο γεγονός ότι ακόμη και η ανθρώπινη νοημοσύνη δεν έχει οριστεί με απόλυτη ακρίβεια από την επιστημονική κοινότητα. Σύμφωνα με τους Russell και Norvig (2016), η Τεχνητή Νοημοσύνη αναφέρεται στη μελέτη και κατασκευή ευφυών πρακτόρων (intelligent agents), δηλαδή συστημάτων που αντιλαμβάνονται το περιβάλλον τους και λαμβάνουν αποφάσεις με στόχο τη μεγιστοποίηση της επιτυχίας τους σε συγκεκριμένες εργασίες. Ο ορισμός αυτός θεωρείται ένας από τους πλέον ολοκληρωμένους, καθώς συνδυάζει την ικανότητα αντίληψης, επεξεργασίας πληροφοριών και λήψης αποφάσεων.

Στη βιβλιογραφία έχουν διατυπωθεί διαφορετικές προσεγγίσεις σχετικά με τον τρόπο κατανόησης της Τεχνητής Νοημοσύνης. Οι Russell και Norvig (2016) ταξινομούν τις προσεγγίσεις της Τεχνητής Νοημοσύνης σε τέσσερις βασικές κατηγορίες: συστήματα που σκέφτονται όπως οι άνθρωποι, συστήματα που ενεργούν όπως οι άνθρωποι, συστήματα που σκέφτονται λογικά και συστήματα που ενεργούν λογικά. Οι δύο πρώτες προσεγγίσεις επιδιώκουν την προσομοίωση της ανθρώπινης σκέψης και συμπεριφοράς, ενώ οι δύο τελευταίες βασίζονται στην έννοια της ορθολογικότητας (rationality). Ως ορθολογική συμπεριφορά νοείται η ικανότητα ενός ευφυούς συστήματος να επιλέγει, μεταξύ των διαθέσιμων εναλλακτικών, την ενέργεια που μεγιστοποιεί την πιθανότητα επίτευξης των προκαθορισμένων στόχων του, αξιοποιώντας τις πληροφορίες που διαθέτει και λαμβάνοντας υπόψη τους περιορισμούς του περιβάλλοντος στο οποίο λειτουργεί (Russell & Norvig, 2016). Επομένως, ένα ορθολογικό σύστημα δεν είναι απαραίτητα εκείνο που μιμείται τον τρόπο σκέψης του ανθρώπου, αλλά εκείνο που λαμβάνει τις πλέον κατάλληλες αποφάσεις με βάση τα διαθέσιμα δεδομένα και το επιδιωκόμενο αποτέλεσμα. Στη σύγχρονη έρευνα, η έννοια του «ορθολογικού πράκτορα» έχει επικρατήσει ως το κυρίαρχο θεωρητικό μοντέλο, καθώς δίνει έμφαση στην αποτελεσματικότητα και στην επίτευξη στόχων ανεξάρτητα από το αν η διαδικασία μιμείται πλήρως τον ανθρώπινο τρόπο σκέψης (Russell & Norvig, 2016).

Η Τεχνητή Νοημοσύνη δεν αποτελεί ενιαία τεχνολογία αλλά ένα σύνολο διαφορετικών μεθόδων και τεχνικών. Σε αυτές περιλαμβάνονται η αναπαράσταση γνώσης, η λογική συλλογιστική, η αναζήτηση λύσεων, η μηχανική μάθηση, τα νευρωνικά δίκτυα, η επεξεργασία φυσικής γλώσσας, η υπολογιστική όραση και τα ευφυή συστήματα λήψης αποφάσεων (Bishop, 2006). Όλες αυτές οι τεχνολογίες έχουν κοινό στόχο την ανάπτυξη συστημάτων που μπορούν να επιδεικνύουν συμπεριφορές οι οποίες παραδοσιακά θεωρούνται χαρακτηριστικά της ανθρώπινης νοημοσύνης.

Σημαντική διάκριση στη θεωρία της Τεχνητής Νοημοσύνης αποτελεί ο διαχωρισμός μεταξύ Ασθενούς Τεχνητής Νοημοσύνης (Weak AI) και Ισχυρής Τεχνητής Νοημοσύνης (Strong AI). Η Ασθενής Τεχνητή Νοημοσύνη αφορά συστήματα τα οποία έχουν σχεδιαστεί για την εκτέλεση συγκεκριμένων εργασιών, όπως η αναγνώριση εικόνων, η μετάφραση κειμένου ή η δημιουργία κώδικα λογισμικού (Bostrom, 2014). Τα περισσότερα σύγχρονα συστήματα AI ανήκουν σε αυτή την κατηγορία. Αντίθετα, η Ισχυρή Τεχνητή Νοημοσύνη αναφέρεται στην υποθετική δυνατότητα δημιουργίας μηχανών που θα διαθέτουν γενική νοημοσύνη αντίστοιχη ή ανώτερη από εκείνη του ανθρώπου, με ικανότητα κατανόησης, αυτοσυνείδησης και αυτόνομης μάθησης σε διαφορετικά γνωστικά πεδία (Bostrom, 2014). Μέχρι σήμερα δεν έχει επιτευχθεί η ανάπτυξη τέτοιων συστημάτων, ωστόσο το ζήτημα παραμένει αντικείμενο έντονης επιστημονικής συζήτησης.

Η σύγχρονη ανάπτυξη της Τεχνητής Νοημοσύνης βασίζεται σε μεγάλο βαθμό στη διαθεσιμότητα μεγάλου όγκου δεδομένων και στη δυνατότητα των συστημάτων να μαθαίνουν από αυτά. Η παραδοσιακή προσέγγιση προγραμματισμού απαιτούσε από τους προγραμματιστές να ορίζουν ρητά όλους τους κανόνες που έπρεπε να ακολουθεί ένα σύστημα. Αντίθετα, τα σύγχρονα συστήματα Τεχνητής Νοημοσύνης μπορούν να ανακαλύπτουν αυτόματα πρότυπα και σχέσεις μέσα από την επεξεργασία μεγάλων συνόλων δεδομένων, προσαρμόζοντας τη συμπεριφορά τους χωρίς να απαιτείται άμεσος επαναπρογραμματισμός (Bishop, 2006). Η δυνατότητα αυτή έχει οδηγήσει στην ανάπτυξη εφαρμογών με υψηλό επίπεδο αυτονομίας και προσαρμοστικότητας.

Η συμβολή της Τεχνητής Νοημοσύνης στον τομέα της ανάπτυξης λογισμικού είναι ιδιαίτερα σημαντική. Παραδοσιακά, η δημιουργία λογισμικού απαιτούσε εκτεταμένη ανθρώπινη εργασία σε όλα τα στάδια ανάπτυξης, από την ανάλυση απαιτήσεων έως τη συντήρηση του τελικού προϊόντος (Sommerville, 2011). Σήμερα, εργαλεία που βασίζονται στην Τεχνητή Νοημοσύνη είναι σε θέση να υποστηρίζουν ή ακόμη και να αυτοματοποιούν πολλές από αυτές τις διαδικασίες. Ειδικότερα, τα συστήματα AI μπορούν να δημιουργούν κώδικα, να εντοπίζουν σφάλματα, να προτείνουν βελτιώσεις, να παράγουν τεκμηρίωση και να υποστηρίζουν τη διαδικασία δοκιμών λογισμικού (Tambon et al., 2025).

Η εμφάνιση μεγάλων γλωσσικών μοντέλων και προηγμένων συστημάτων παραγωγικής Τεχνητής Νοημοσύνης έχει επιταχύνει περαιτέρω τη διείσδυση της AI στη μηχανική λογισμικού. Εφαρμογές όπως το ChatGPT και το GitHub Copilot είναι πλέον σε θέση να κατανοούν εντολές φυσικής γλώσσας και να μετατρέπουν τις απαιτήσεις των χρηστών σε λειτουργικό κώδικα (Chen et al., 2021; Nijkamp et al., 2022).

Η εξέλιξη αυτή μεταβάλλει σταδιακά τον τρόπο εργασίας των προγραμματιστών και δημιουργεί νέες προοπτικές για τη βελτίωση της παραγωγικότητας και της ποιότητας του λογισμικού (Tambon et al., 2025). Παράλληλα, όμως, εγείρει σημαντικούς επιστημονικούς, κοινωνικούς και περιβαλλοντικούς προβληματισμούς, οι οποίοι απασχολούν ολοένα και περισσότερο την ερευνητική κοινότητα (Bostrom, 2014). Ένας από τους βασικότερους προβληματισμούς αφορά τον μελλοντικό ρόλο των επαγγελματιών της ανάπτυξης λογισμικού. Αν και τα σύγχρονα εργαλεία Τεχνητής Νοημοσύνης μπορούν να αυτοματοποιούν σημαντικό μέρος της συγγραφής κώδικα, του εντοπισμού σφαλμάτων και της παραγωγής τεκμηρίωσης, παραμένει ανοικτό το ερώτημα κατά πόσο θα λειτουργήσουν ως υποστηρικτικά εργαλεία για τους προγραμματιστές ή εάν, σε βάθος χρόνου, θα οδηγήσουν στον μετασχηματισμό ή ακόμη και στη μερική αντικατάσταση ορισμένων επαγγελματικών ρόλων (Russell & Norvig, 2016). Παράλληλα, αναδεικνύεται ένας δεύτερος, λιγότερο προβεβλημένος αλλά ιδιαίτερα σημαντικός προβληματισμός, που αφορά το υψηλό ενεργειακό κόστος της εκπαίδευσης και λειτουργίας των μεγάλων μοντέλων Τεχνητής Νοημοσύνης. Η ανάπτυξη και η λειτουργία τέτοιων μοντέλων απαιτούν υποδομές μεγάλης υπολογιστικής ισχύος, οι οποίες καταναλώνουν σημαντικές ποσότητες ηλεκτρικής ενέργειας και συμβάλλουν στην αύξηση του περιβαλλοντικού αποτυπώματος των κέντρων δεδομένων (Bender et al., 2021). Κατά συνέπεια, η μελλοντική εξέλιξη της Τεχνητής Νοημοσύνης δεν θα πρέπει να αξιολογείται μόνο με κριτήρια τεχνολογικής αποτελεσματικότητας, αλλά και υπό το πρίσμα της βιωσιμότητας, της ενεργειακής αποδοτικότητας, της κοινωνικής υπευθυνότητας και των επιπτώσεών της στην αγορά εργασίας.

1.2 Ιστορική εξέλιξη της Τεχνητής Νοημοσύνης

Η ιστορία της Τεχνητής Νοημοσύνης αποτελεί μία από τις πλέον ενδιαφέρουσες διαδρομές στην εξέλιξη της επιστήμης της Πληροφορικής. Η προσπάθεια δημιουργίας μηχανών που θα μπορούσαν να προσομοιώσουν ανθρώπινες νοητικές λειτουργίες δεν αποτελεί αποκλειστικά προϊόν της σύγχρονης τεχνολογικής εποχής, αλλά συνδέεται με φιλοσοφικούς προβληματισμούς και επιστημονικές αναζητήσεις που εμφανίζονται ήδη από την αρχαιότητα. Ωστόσο, η Τεχνητή Νοημοσύνη ως οργανωμένο επιστημονικό πεδίο διαμορφώθηκε κατά τον 20ό αιώνα, όταν η ανάπτυξη των ηλεκτρονικών υπολογιστών δημιούργησε τις προϋποθέσεις για την υλοποίηση θεωρητικών ιδεών σχετικά με τη μηχανική σκέψη και την αυτοματοποίηση της λήψης αποφάσεων (Nilsson, 2009).

Οι πρώτες θεωρητικές βάσεις της Τεχνητής Νοημοσύνης συνδέονται άμεσα με το έργο του μαθηματικού και επιστήμονα υπολογιστών Alan Turing. Το 1950 ο Turing δημοσίευσε το ιστορικό άρθρο «Computing Machinery and Intelligence», στο οποίο διατύπωσε το ερώτημα «Μπορούν οι μηχανές να σκεφτούν;» (Turing, 1950). Η σημασία του έργου του δεν περιορίζεται μόνο στη θεωρητική διερεύνηση της μηχανικής νοημοσύνης, αλλά επεκτείνεται και στη διαμόρφωση ενός πρακτικού τρόπου αξιολόγησης της ευφυΐας των υπολογιστικών συστημάτων μέσω του γνωστού Τεστ Turing. Σύμφωνα με αυτή τη δοκιμασία, ένα υπολογιστικό σύστημα θεωρείται ότι παρουσιάζει ευφυή συμπεριφορά όταν ένας άνθρωπος δεν μπορεί να διακρίνει αν συνομιλεί με άνθρωπο ή με μηχανή (Turing, 1950). Η προσέγγιση αυτή επηρέασε βαθιά τη μετέπειτα πορεία του πεδίου και εξακολουθεί να αποτελεί σημείο αναφοράς στη συζήτηση για τη φύση της Τεχνητής Νοημοσύνης.

Η επίσημη γέννηση της Τεχνητής Νοημοσύνης ως επιστημονικού κλάδου πραγματοποιήθηκε το 1956 κατά τη διάρκεια του θερινού ερευνητικού προγράμματος στο Dartmouth College των Ηνωμένων Πολιτειών. Το πρόγραμμα οργανώθηκε από τους John McCarthy, Marvin Minsky, Nathaniel Rochester και Claude Shannon, οι οποίοι εισήγαγαν για πρώτη φορά τον όρο «Artificial Intelligence» (McCarthy et al., 1955). Οι ερευνητές πίστευαν ότι οι γνωστικές λειτουργίες του ανθρώπου, όπως η μάθηση, η λογική σκέψη και η επίλυση προβλημάτων, θα μπορούσαν να αναπαρασταθούν μέσω αλγορίθμων και να υλοποιηθούν σε υπολογιστικά συστήματα. Η αισιοδοξία εκείνης της περιόδου ήταν ιδιαίτερα έντονη, καθώς πολλοί επιστήμονες θεωρούσαν ότι η δημιουργία πραγματικά ευφυών μηχανών θα ήταν εφικτή μέσα σε λίγες δεκαετίες (Nilsson, 2009).

Κατά τις δεκαετίες του 1950 και του 1960 αναπτύχθηκαν τα πρώτα προγράμματα που θεωρήθηκαν παραδείγματα Τεχνητής Νοημοσύνης. Ένα από τα σημαντικότερα ήταν το Logic Theorist, το οποίο σχεδιάστηκε από τους Allen Newell και Herbert Simon και είχε τη δυνατότητα να αποδεικνύει μαθηματικά θεωρήματα χρησιμοποιώντας λογικούς κανόνες (Nilsson, 2009). Το

πρόγραμμα αυτό θεωρήθηκε επαναστατικό, καθώς έδειξε ότι οι υπολογιστές μπορούσαν να επιλύουν προβλήματα που μέχρι τότε θεωρούνταν αποκλειστικά ανθρώπινη δραστηριότητα. Παράλληλα, αναπτύχθηκαν συστήματα επίλυσης προβλημάτων και πρώιμα προγράμματα επεξεργασίας φυσικής γλώσσας, τα οποία ενίσχυσαν την αισιοδοξία για το μέλλον του πεδίου.

Σημαντικό ορόσημο στην εξέλιξη της Τεχνητής Νοημοσύνης αποτέλεσε το έργο του Frank Rosenblatt, ο οποίος το 1957 παρουσίασε το Perceptron, ένα από τα πρώτα τεχνητά νευρωνικά δίκτυα (Rosenblatt, 1957). Το Perceptron σχεδιάστηκε με βάση τη λειτουργία των βιολογικών νευρώνων και είχε τη δυνατότητα να μαθαίνει από παραδείγματα. Η ανάπτυξή του δημιούργησε μεγάλες προσδοκίες σχετικά με την ικανότητα των μηχανών να αποκτούν γνώση και να προσαρμόζονται σε νέα δεδομένα. Ωστόσο, οι περιορισμένες δυνατότητες των υπολογιστικών συστημάτων της εποχής και οι θεωρητικοί περιορισμοί που επισημάνθηκαν αργότερα από τους Minsky και Papert (1969) οδήγησαν σε σημαντική μείωση του ενδιαφέροντος για τα νευρωνικά δίκτυα κατά τις επόμενες δεκαετίες.

Η περίοδος από τα μέσα της δεκαετίας του 1970 έως τις αρχές της δεκαετίας του 1980 είναι γνωστή ως ο πρώτος «χειμώνας της Τεχνητής Νοημοσύνης» (AI Winter). Κατά τη διάρκεια αυτής της περιόδου, οι υπερβολικές προσδοκίες που είχαν δημιουργηθεί δεν επιβεβαιώθηκαν στην πράξη, με αποτέλεσμα τη σημαντική μείωση της χρηματοδότησης και του ερευνητικού ενδιαφέροντος (Nilsson, 2009). Οι υπολογιστικές δυνατότητες της εποχής δεν ήταν επαρκείς για την υλοποίηση των φιλόδοξων στόχων που είχαν τεθεί, ενώ πολλά συστήματα αποδείχθηκαν αδύναμα όταν κλήθηκαν να λειτουργήσουν σε πραγματικά και σύνθετα περιβάλλοντα.

Η αναβίωση της Τεχνητής Νοημοσύνης ξεκίνησε κατά τη δεκαετία του 1980 με την εμφάνιση των έμπειρων συστημάτων (Expert Systems), βασισμένων σε γλώσσες προγραμματισμού 5ης γενιάς. Τα συστήματα αυτά βασίζονταν σε μεγάλες βάσεις γνώσης και σε κανόνες λογικής συμπερασματολογίας, επιτρέποντας την επίλυση εξειδικευμένων προβλημάτων σε τομείς όπως η ιατρική διάγνωση, η βιομηχανική παραγωγή και η χρηματοοικονομική ανάλυση (Russell & Norvig, 2016). Η εμπορική επιτυχία ορισμένων έμπειρων συστημάτων οδήγησε σε νέα κύματα επενδύσεων και αναζωπύρωσε το ενδιαφέρον για την Τεχνητή Νοημοσύνη. Παρά τα σημαντικά πλεονεκτήματά τους, τα συστήματα αυτά παρουσίαζαν περιορισμούς ως προς την προσαρμοστικότητα και την ικανότητα μάθησης, καθώς βασίζονταν σε κανόνες που είχαν καθοριστεί εκ των προτέρων από ανθρώπους ειδικούς.

Κατά τη δεκαετία του 1990 η έμφαση μετατοπίστηκε από τα συστήματα κανόνων στις τεχνικές μηχανικής μάθησης. Οι ερευνητές άρχισαν να αναπτύσσουν αλγορίθμους που μπορούσαν να εξάγουν γνώση απευθείας από δεδομένα, χωρίς να απαιτείται η ρητή διατύπωση κανόνων από τον προγραμματιστή (Bishop, 2006). Η αλλαγή αυτή σηματοδότησε μία από τις σημαντικότερες μεταβάσεις στην ιστορία της Τεχνητής Νοημοσύνης, καθώς έθεσε τις βάσεις για τα σύγχρονα συστήματα μάθησης. Κατά την ίδια περίοδο σημειώθηκαν και σημαντικά επιτεύγματα, όπως η νίκη του υπολογιστικού συστήματος Deep Blue της IBM απέναντι στον παγκόσμιο πρωταθλητή σκακιού Garry Kasparov το 1997, γεγονός που θεωρήθηκε σημαντικό ορόσημο για την πρόοδο της υπολογιστικής νοημοσύνης (Russell & Norvig, 2016).

Η πραγματική έκρηξη της Τεχνητής Νοημοσύνης πραγματοποιήθηκε μετά το 2010, όταν η αύξηση της υπολογιστικής ισχύος, η διάδοση των υπολογιστικών νεφών (cloud computing) και η διαθεσιμότητα τεράστιων ποσοτήτων δεδομένων επέτρεψαν την ανάπτυξη προηγμένων μοντέλων βαθιάς μάθησης (Deep Learning) (Bishop, 2006). Τα πολυεπίπεδα νευρωνικά δίκτυα κατέστησαν δυνατή την επίτευξη εξαιρετικών επιδόσεων σε τομείς όπως η αναγνώριση εικόνας, η αναγνώριση φωνής, η επεξεργασία φυσικής γλώσσας και η αυτόνομη πλοήγηση οχημάτων. Η εξέλιξη αυτή επανέφερε την Τεχνητή Νοημοσύνη στο επίκεντρο της επιστημονικής έρευνας και των επιχειρηματικών επενδύσεων.

Τα τελευταία χρόνια, η ανάπτυξη των Μεγάλων Γλωσσικών Μοντέλων (Large Language Models - LLMs) έχει δημιουργήσει μια νέα εποχή για την Τεχνητή Νοημοσύνη. Μοντέλα όπως το GPT της OpenAI μπορούν να παράγουν κείμενο, να απαντούν σε ερωτήσεις, να μεταφράζουν γλώσσες και να δημιουργούν κώδικα λογισμικού με υψηλό βαθμό ακρίβειας (Chen et al., 2021). Η εμφάνιση εφαρμογών όπως το ChatGPT και το GitHub Copilot έχει μεταβάλει ουσιαστικά τον τρόπο με τον οποίο οι άνθρωποι αλληλεπιδρούν με τα πληροφοριακά συστήματα και αναπτύσσουν λογισμικό (Tambon et al., 2025).

1.3 Μηχανική Μάθηση και Γενετική Τεχνητή Νοημοσύνη

Η ραγδαία ανάπτυξη της Τεχνητής Νοημοσύνης κατά τις τελευταίες δεκαετίες συνδέεται άμεσα με την πρόοδο που σημειώθηκε στον τομέα της Μηχανικής Μάθησης (Machine Learning) και, πιο πρόσφατα, της Γενετικής Τεχνητής Νοημοσύνης (Generative Artificial Intelligence). Οι δύο αυτές τεχνολογικές κατηγορίες αποτελούν σήμερα τον πυρήνα πολλών σύγχρονων εφαρμογών τεχνητής νοημοσύνης και έχουν επηρεάσει καθοριστικά τον τρόπο με τον οποίο αναπτύσσονται πληροφοριακά συστήματα και εφαρμογές λογισμικού. Η κατανόηση των βασικών αρχών λειτουργίας τους είναι απαραίτητη για την κατανόηση της σύγχρονης εξέλιξης της μηχανικής λογισμικού και των νέων εργαλείων που χρησιμοποιούνται στην παραγωγή κώδικα και την αυτοματοποίηση διαδικασιών ανάπτυξης (Bishop, 2006).

Η Μηχανική Μάθηση αποτελεί υποπεδίο της Τεχνητής Νοημοσύνης και επικεντρώνεται στην ανάπτυξη αλγορίθμων που επιτρέπουν στους υπολογιστές να μαθαίνουν από δεδομένα χωρίς να απαιτείται ρητός προγραμματισμός όλων των κανόνων λειτουργίας τους (Russell & Norvig, 2016). Σε αντίθεση με τις παραδοσιακές μεθόδους ανάπτυξης λογισμικού, όπου ο προγραμματιστής καθορίζει λεπτομερώς κάθε πιθανή συνθήκη και απόφαση που πρέπει να λάβει το σύστημα, η Μηχανική Μάθηση επιτρέπει στα συστήματα να αναγνωρίζουν πρότυπα, να εξάγουν συμπεράσματα και να βελτιώνουν την απόδοσή τους μέσα από την ανάλυση δεδομένων (Bishop, 2006). Η προσέγγιση αυτή έχει αποδειχθεί ιδιαίτερα αποτελεσματική σε προβλήματα όπου η πολυπλοκότητα των δεδομένων καθιστά αδύνατη τη χειροκίνητη διατύπωση όλων των απαιτούμενων κανόνων.

Η βασική αρχή της Μηχανικής Μάθησης στηρίζεται στη δημιουργία μαθηματικών μοντέλων που εκπαιδεύονται χρησιμοποιώντας ιστορικά δεδομένα. Κατά τη διαδικασία της εκπαίδευσης, ο αλγόριθμος αναλύει παραδείγματα και προσαρμόζει τις εσωτερικές του παραμέτρους, με στόχο να μπορεί να προβλέπει ή να ταξινομεί σωστά νέα δεδομένα που δεν έχει συναντήσει στο παρελθόν (Hastie, Tibshirani & Friedman, 2008). Με τον τρόπο αυτό, το σύστημα αποκτά τη δυνατότητα γενίκευσης, δηλαδή εφαρμογής της γνώσης που έχει αποκτήσει σε νέες καταστάσεις.

Η βιβλιογραφία διακρίνει τρεις βασικές κατηγορίες Μηχανικής Μάθησης. Η πρώτη είναι η επιβλεπόμενη μάθηση (Supervised Learning), κατά την οποία το σύστημα εκπαιδεύεται χρησιμοποιώντας δεδομένα που συνοδεύονται από γνωστές απαντήσεις ή ετικέτες. Στόχος είναι η πρόβλεψη της σωστής εξόδου για νέα δεδομένα με βάση τα παραδείγματα που έχουν προηγουμένως παρασχεθεί (Bishop, 2006).

Η δεύτερη κατηγορία είναι η μη επιβλεπόμενη μάθηση (Unsupervised Learning), στην οποία το σύστημα δεν διαθέτει προκαθορισμένες ετικέτες ή σωστές απαντήσεις. Αντίθετα, προσπαθεί να ανακαλύψει μόνο του δομές, πρότυπα ή ομάδες μέσα στα δεδομένα (Bishop, 2006). Οι τεχνικές αυτές χρησιμοποιούνται ευρέως σε εφαρμογές ανάλυσης δεδομένων, κατηγοριοποίησης πελατών, ανίχνευσης ανωμαλιών και εξόρυξης γνώσης.

Η τρίτη κατηγορία είναι η ενισχυτική μάθηση (Reinforcement Learning), όπου ένας πράκτορας αλληλεπιδρά με το περιβάλλον του και μαθαίνει μέσω ανταμοιβών και ποινών. Ο στόχος είναι η ανάπτυξη στρατηγικών που μεγιστοποιούν το συνολικό όφελος μέσα από επαναλαμβανόμενες δοκιμές και εμπειρίες (Russell & Norvig, 2016). Η προσέγγιση αυτή έχει χρησιμοποιηθεί με επιτυχία σε εφαρμογές όπως η ρομποτική, τα αυτόνομα οχήματα και τα συστήματα λήψης αποφάσεων.

Καθοριστική συμβολή στην εξέλιξη της Μηχανικής Μάθησης είχε η ανάπτυξη των τεχνητών νευρωνικών δικτύων. Τα δίκτυα αυτά σχεδιάστηκαν με βάση τη δομή και τη λειτουργία του ανθρώπινου εγκεφάλου, αποτελούμενα από μεγάλο αριθμό τεχνητών νευρώνων που συνεργάζονται για την επεξεργασία πληροφοριών (Rosenblatt, 1957). Παρότι τα πρώτα νευρωνικά δίκτυα παρουσίαζαν σημαντικούς περιορισμούς, η πρόοδος της υπολογιστικής ισχύος και η ανάπτυξη νέων τεχνικών εκπαίδευσης οδήγησαν στη δημιουργία βαθιών νευρωνικών δικτύων (Deep Neural Networks), τα οποία αποτελούν τη βάση της σύγχρονης βαθιάς μάθησης (Deep Learning) (Goodfellow, Bengio & Courville, 2016).

Η βαθιά μάθηση αποτελεί σήμερα μία από τις σημαντικότερες τεχνολογίες της Τεχνητής Νοημοσύνης. Τα μοντέλα βαθιάς μάθησης χρησιμοποιούν πολλαπλά επίπεδα επεξεργασίας δεδομένων, επιτρέποντας την αυτόματη εξαγωγή σύνθετων χαρακτηριστικών και προτύπων χωρίς την ανάγκη χειροκίνητου σχεδιασμού τους από τον άνθρωπο (Goodfellow et al., 2016). Χάρη σε αυτή τη δυνατότητα, τα συστήματα βαθιάς μάθησης έχουν επιτύχει εντυπωσιακά

αποτελέσματα στην αναγνώριση εικόνων, στην επεξεργασία φυσικής γλώσσας, στην αναγνώριση ομιλίας και σε πολλές άλλες εφαρμογές.

Μία από τις σημαντικότερες εξελίξεις των τελευταίων ετών είναι η εμφάνιση της Γενετικής Τεχνητής Νοημοσύνης (Generative Artificial Intelligence). Σε αντίθεση με τα παραδοσιακά συστήματα Μηχανικής Μάθησης, τα οποία εστιάζουν κυρίως στην ταξινόμηση ή πρόβλεψη δεδομένων, τα γενετικά μοντέλα έχουν τη δυνατότητα να δημιουργούν νέο περιεχόμενο, που δεν υπήρχε προηγουμένως στα δεδομένα εκπαίδευσής τους (Bostrom, 2014). Το περιεχόμενο αυτό μπορεί να περιλαμβάνει κείμενο, εικόνες, μουσική, βίντεο, λογισμικό ή άλλες μορφές ψηφιακού υλικού.

Η ανάπτυξη της Γενετικής Τεχνητής Νοημοσύνης βασίζεται κυρίως στα Μεγάλα Γλωσσικά Μοντέλα (Large Language Models - LLMs). Τα μοντέλα αυτά εκπαιδεύονται σε τεράστιες συλλογές κειμένων και μαθαίνουν να προβλέπουν την επόμενη λέξη ή ακολουθία λέξεων μέσα σε ένα κείμενο (Chen et al., 2021). Μέσα από αυτή τη διαδικασία αποκτούν τη δυνατότητα παραγωγής συνεκτικού και νοηματικά ολοκληρωμένου λόγου, γεγονός που τα καθιστά ιδιαίτερα χρήσιμα σε εφαρμογές όπως οι ψηφιακοί βοηθοί, τα συστήματα συνομιλίας και τα εργαλεία δημιουργίας λογισμικού.

Παρά τα σημαντικά πλεονεκτήματα, η χρήση της Γενετικής Τεχνητής Νοημοσύνης συνοδεύεται και από προκλήσεις. Ένα από τα σημαντικότερα ζητήματα αφορά την αξιοπιστία του παραγόμενου περιεχομένου. Τα γενετικά μοντέλα ενδέχεται να παράγουν λανθασμένες πληροφορίες ή μη βέλτιστο κώδικα, δημιουργώντας προβλήματα λειτουργικότητας και ασφάλειας (Pearce et al., 2022).

Η Μηχανική Μάθηση και η Γενετική Τεχνητή Νοημοσύνη αποτελούν τις δύο σημαντικότερες τεχνολογικές εξελίξεις στο σύγχρονο πεδίο της Τεχνητής Νοημοσύνης. Η πρώτη επιτρέπει στα συστήματα να μαθαίνουν από δεδομένα και να βελτιώνουν την απόδοσή τους, ενώ η δεύτερη επεκτείνει τις δυνατότητές τους δίνοντάς τους τη δυνατότητα δημιουργίας νέου περιεχομένου. Η συνεχής εξέλιξή τους επηρεάζει ήδη σημαντικά τη μηχανική λογισμικού και αναμένεται να διαμορφώσει το μέλλον της ανάπτυξης εφαρμογών τα επόμενα χρόνια.

Κεφάλαιο 2: Η Δημιουργία Λογισμικού στη Σύγχρονη Εποχή

2.1 Βασικά στάδια ανάπτυξης λογισμικού

Η ανάπτυξη λογισμικού αποτελεί μία οργανωμένη και συστηματική διαδικασία, μέσω της οποίας οι απαιτήσεις των χρηστών μετατρέπονται σε λειτουργικές εφαρμογές και πληροφοριακά συστήματα. Πρόκειται για έναν από τους σημαντικότερους κλάδους της επιστήμης της Πληροφορικής, καθώς το λογισμικό αποτελεί βασικό συστατικό σχεδόν κάθε σύγχρονης τεχνολογικής υποδομής, από τις εφαρμογές κινητών τηλεφώνων και τις διαδικτυακές πλατφόρμες μέχρι τα πληροφοριακά συστήματα επιχειρήσεων και τα ενσωματωμένα συστήματα (embedded systems) που χρησιμοποιούνται στη βιομηχανία και στις μεταφορές (Sommerville, 2011). Η επιτυχής ανάπτυξη λογισμικού προϋποθέτει τον συνδυασμό τεχνικών γνώσεων, κατάλληλων μεθοδολογιών, αποτελεσματικής διαχείρισης έργων και συνεχούς συνεργασίας μεταξύ προγραμματιστών, αναλυτών, σχεδιαστών και τελικών χρηστών (Pressman & Maxim, 2020).

Η πολυπλοκότητα των σύγχρονων πληροφοριακών συστημάτων κατέστησε αναγκαία την υιοθέτηση ενός οργανωμένου πλαισίου ανάπτυξης, γνωστού ως Κύκλος Ζωής Ανάπτυξης Λογισμικού (Software Development Life Cycle - SDLC). Ο κύκλος αυτός περιγράφει τα βασικά στάδια που ακολουθούνται από τη σύλληψη μιας ιδέας έως την παράδοση και τη συντήρηση του τελικού προϊόντος, συμβάλλοντας στη μείωση των λαθών, στον καλύτερο προγραμματισμό των εργασιών και στη διασφάλιση της ποιότητας του λογισμικού (ISO/IEC/IEEE 12207, 2017). Παρότι υπάρχουν διαφορετικές μεθοδολογίες ανάπτυξης, όπως το μοντέλο Waterfall, οι ευέλικτες μεθοδολογίες (Agile) και το DevOps, τα βασικά στάδια του κύκλου ζωής παραμένουν σε μεγάλο βαθμό κοινά (Sommerville, 2011).

Το πρώτο στάδιο είναι η ανάλυση και καταγραφή των απαιτήσεων. Κατά τη φάση αυτή προσδιορίζονται οι ανάγκες του οργανισμού ή των τελικών χρηστών, καθώς και οι λειτουργικές και μη λειτουργικές απαιτήσεις που πρέπει να ικανοποιεί το υπό ανάπτυξη σύστημα. Οι λειτουργικές απαιτήσεις περιγράφουν τις υπηρεσίες και τις δυνατότητες που θα παρέχει η

εφαρμογή, ενώ οι μη λειτουργικές αφορούν χαρακτηριστικά όπως η ασφάλεια, η αξιοπιστία, η απόδοση, η επεκτασιμότητα και η ευχρηστία (Pressman & Maxim, 2020). Η σωστή ανάλυση απαιτήσεων θεωρείται καθοριστική για την επιτυχία ενός έργου, καθώς λανθασμένες ή ελλιπείς προδιαγραφές συχνά οδηγούν σε αυξημένο κόστος ανάπτυξης και σε προβλήματα που είναι δύσκολο να διορθωθούν στα επόμενα στάδια (Brooks, 1995).

Ακολουθεί το στάδιο του σχεδιασμού του συστήματος (System Design), κατά το οποίο μετατρέπονται οι απαιτήσεις σε μία ολοκληρωμένη αρχιτεκτονική λύση. Στο στάδιο αυτό σχεδιάζονται η δομή της εφαρμογής, η αρχιτεκτονική του λογισμικού, οι βάσεις δεδομένων, οι διεπαφές χρήστη (User Interfaces), οι μηχανισμοί επικοινωνίας μεταξύ των υποσυστημάτων και οι τεχνολογίες που θα χρησιμοποιηθούν κατά την υλοποίηση. Ένας σωστός σχεδιασμός συμβάλλει στην καλύτερη συντηρησιμότητα, στην επεκτασιμότητα και στην ασφάλεια του τελικού λογισμικού, ενώ μειώνει τον κίνδυνο εμφάνισης τεχνικού χρέους (technical debt) κατά τη διάρκεια της ανάπτυξης. Με τον όρο τεχνικό χρέος περιγράφονται οι πρόσθετες μελλοντικές επιβαρύνσεις που δημιουργούνται όταν, για λόγους ταχύτητας ή ευκολίας, επιλέγονται προσωρινές ή λιγότερο κατάλληλες τεχνικές λύσεις αντί μιας περισσότερο ολοκληρωμένης προσέγγισης. Οι επιλογές αυτές μπορεί να διευκολύνουν την άμεση ανάπτυξη, αλλά συχνά αυξάνουν την ανάγκη για μελλοντικές διορθώσεις, ανασχεδιασμό και πρόσθετη συντήρηση του λογισμικού (Martin, 2008).

Το επόμενο στάδιο είναι η υλοποίηση ή ανάπτυξη κώδικα (Implementation ή Coding). Στη φάση αυτή οι προγραμματιστές μετατρέπουν τις τεχνικές προδιαγραφές σε εκτελέσιμο κώδικα χρησιμοποιώντας γλώσσες προγραμματισμού, όπως Java, Python, C#, JavaScript ή άλλες, ανάλογα με τις απαιτήσεις του έργου. Παράλληλα, χρησιμοποιούνται ολοκληρωμένα περιβάλλοντα ανάπτυξης (Integrated Development Environments - IDEs), συστήματα ελέγχου εκδόσεων όπως το Git, καθώς και διάφορα εργαλεία αυτοματοποίησης που διευκολύνουν τη συνεργασία μεταξύ των μελών της ομάδας ανάπτυξης (Pressman & Maxim, 2020).

Μετά την ολοκλήρωση της υλοποίησης ακολουθεί η φάση του ελέγχου (Testing). Σκοπός της είναι να διαπιστωθεί ότι το λογισμικό λειτουργεί σύμφωνα με τις προδιαγραφές και ότι δεν παρουσιάζει σφάλματα που θα μπορούσαν να επηρεάσουν τη λειτουργικότητα ή την ασφάλειά του. Οι δοκιμές πραγματοποιούνται σε διαφορετικά επίπεδα και περιλαμβάνουν δοκιμές μονάδας (Unit Testing), δοκιμές ολοκλήρωσης (Integration Testing), δοκιμές συστήματος (System Testing), δοκιμές αποδοχής από τον χρήστη (User Acceptance Testing - UAT) και δοκιμές απόδοσης ή ασφάλειας, ανάλογα με τη φύση της εφαρμογής (Sommerville, 2011). Η έγκαιρη ανίχνευση σφαλμάτων μειώνει σημαντικά το κόστος διόρθωσής τους και αυξάνει την αξιοπιστία του τελικού προϊόντος.

Το επόμενο στάδιο είναι η εγκατάσταση και διάθεση του λογισμικού (Deployment). Αφού ολοκληρωθούν επιτυχώς οι δοκιμές, η εφαρμογή εγκαθίσταται στο πραγματικό περιβάλλον λειτουργίας, όπου καθίσταται διαθέσιμη στους τελικούς χρήστες. Στις σύγχρονες πρακτικές ανάπτυξης η διαδικασία αυτή πραγματοποιείται συχνά μέσω αυτοματοποιημένων μηχανισμών συνεχούς ολοκλήρωσης και συνεχούς διάθεσης (Continuous Integration - Continuous Deployment, CI/CD), οι οποίοι επιτρέπουν την ταχεία και ασφαλή δημοσίευση νέων εκδόσεων του λογισμικού με ελάχιστη ανθρώπινη παρέμβαση (Humble & Farley, 2010).

Η ολοκλήρωση της εγκατάστασης δεν σηματοδοτεί και το τέλος του κύκλου ζωής του λογισμικού. Αντίθετα, ακολουθεί η φάση της συντήρησης (Maintenance), η οποία αποτελεί συχνά το μεγαλύτερο μέρος του συνολικού κόστους ενός έργου λογισμικού. Κατά τη διάρκεια της λειτουργίας μιας εφαρμογής προκύπτουν ανάγκες για διόρθωση σφαλμάτων, βελτίωση των επιδόσεων, ενσωμάτωση νέων λειτουργιών, προσαρμογή σε αλλαγές του τεχνολογικού περιβάλλοντος και αντιμετώπιση νέων κινδύνων ασφάλειας (Pressman & Maxim, 2020). Η συνεχής συντήρηση διασφαλίζει ότι το λογισμικό παραμένει λειτουργικό, ασφαλές και συμβατό με τις μεταβαλλόμενες ανάγκες των χρηστών και των οργανισμών.

Παρότι τα στάδια του κύκλου ζωής παρουσιάζονται συνήθως με διαδοχική σειρά, στη σύγχρονη ανάπτυξη λογισμικού εφαρμόζονται κυρίως επαναληπτικές και ευέλικτες προσεγγίσεις. Μεθοδολογίες όπως η Agile και το Scrum επιτρέπουν την ανάπτυξη του λογισμικού σε διαδοχικούς κύκλους μικρής διάρκειας, κατά τους οποίους πραγματοποιούνται συνεχείς βελτιώσεις και ενσωματώνονται τα σχόλια των χρηστών. Με τον τρόπο αυτό οι ομάδες ανάπτυξης ανταποκρίνονται αποτελεσματικότερα στις αλλαγές των απαιτήσεων και μειώνουν τον κίνδυνο αποτυχίας μεγάλων έργων πληροφορικής (Beck et al., 2001).

2.2 Σύγχρονες τεχνολογίες ανάπτυξης

Η ανάπτυξη λογισμικού έχει εξελιχθεί σημαντικά τις τελευταίες δεκαετίες, ακολουθώντας τις ραγδαίες τεχνολογικές εξελίξεις στον χώρο της Πληροφορικής και ανταποκρινόμενη στις ολοένα αυξανόμενες απαιτήσεις των οργανισμών και των χρηστών. Οι σύγχρονες εφαρμογές χαρακτηρίζονται από υψηλό βαθμό πολυπλοκότητας, μεγάλη κλίμακα λειτουργίας και ανάγκη για συνεχή διαθεσιμότητα, ασφάλεια και επεκτασιμότητα. Ως αποτέλεσμα, οι παραδοσιακές προσεγγίσεις ανάπτυξης λογισμικού έχουν αντικατασταθεί ή συμπληρωθεί από νέες τεχνολογίες, εργαλεία και μεθοδολογίες που επιτρέπουν την ταχύτερη, ποιοτικότερη και αποτελεσματικότερη παραγωγή εφαρμογών (Pressman & Maxim, 2020).

Μία από τις σημαντικότερες εξελίξεις αποτελεί η ευρεία υιοθέτηση του αντικειμενοστρεφούς προγραμματισμού (Object-Oriented Programming - OOP). Η προσέγγιση αυτή βασίζεται στη μοντελοποίηση του λογισμικού μέσω αντικειμένων, τα οποία συνδυάζουν δεδομένα και λειτουργίες σε μία ενιαία δομή. Οι βασικές αρχές του αντικειμενοστρεφούς προγραμματισμού, όπως η ενθυλάκωση (encapsulation), η κληρονομικότητα (inheritance), η πολυμορφία (polymorphism) και η αφαίρεση (abstraction), συμβάλλουν στη δημιουργία λογισμικού που είναι ευκολότερο στη συντήρηση, την επαναχρησιμοποίηση και την επέκταση (Sommerville, 2011). Γλώσσες προγραμματισμού όπως η C++, η Java, η C# και η Python αξιοποιούν σε μεγάλο βαθμό τις αρχές αυτές και χρησιμοποιούνται ευρέως τόσο στην ανάπτυξη επιχειρησιακών εφαρμογών όσο και στην ανάπτυξη εφαρμογών Τεχνητής Νοημοσύνης.

Εξίσου σημαντική είναι η χρήση πλαισίων ανάπτυξης (frameworks) και βιβλιοθηκών λογισμικού (libraries), τα οποία επιτρέπουν στους προγραμματιστές να αξιοποιούν έτοιμες λειτουργίες αντί να αναπτύσσουν κάθε στοιχείο από την αρχή. Frameworks όπως το Spring Boot, το .NET, το Django και το React προσφέρουν προκαθορισμένες δομές ανάπτυξης, μειώνοντας τον χρόνο υλοποίησης και αυξάνοντας την ποιότητα και την ομοιομορφία του παραγόμενου κώδικα (Pressman & Maxim, 2020). Παράλληλα, η αξιοποίηση βιβλιοθηκών ανοικτού λογισμικού ενισχύει τη συνεργασία της παγκόσμιας κοινότητας προγραμματιστών και επιταχύνει την τεχνολογική καινοτομία.

Σημαντική θέση στη σύγχρονη ανάπτυξη λογισμικού κατέχουν επίσης οι Διεπαφές Προγραμματισμού Εφαρμογών (Application Programming Interfaces - APIs). Τα APIs επιτρέπουν την ασφαλή και τυποποιημένη επικοινωνία μεταξύ διαφορετικών εφαρμογών και πληροφοριακών συστημάτων, διευκολύνοντας την ανταλλαγή δεδομένων και τη διαλειτουργικότητα. Μέσω των APIs, οι προγραμματιστές μπορούν να ενσωματώνουν υπηρεσίες τρίτων, όπως συστήματα πληρωμών, χάρτες, υπηρεσίες ηλεκτρονικού ταχυδρομείου ή ακόμη και υπηρεσίες Τεχνητής Νοημοσύνης, χωρίς να απαιτείται η ανάπτυξη των αντίστοιχων λειτουργιών από την αρχή (Fielding, 2000).

Η ανάπτυξη του υπολογιστικού νέφους (Cloud Computing) έχει μεταβάλει ριζικά τον τρόπο σχεδίασης και λειτουργίας των εφαρμογών. Αντί οι οργανισμοί να επενδύουν σε ιδιόκτητες υποδομές πληροφορικής, μπορούν πλέον να αξιοποιούν υπηρεσίες που παρέχονται μέσω πλατφορμών όπως το Amazon Web Services (AWS), το Microsoft Azure και το Google Cloud Platform. Οι υπηρεσίες αυτές προσφέρουν υψηλή διαθεσιμότητα, δυνατότητα δυναμικής κλιμάκωσης των πόρων, αυξημένη αξιοπιστία και σημαντική μείωση του λειτουργικού κόστους, ενώ παράλληλα διευκολύνουν την ανάπτυξη εφαρμογών που εξυπηρετούν μεγάλο αριθμό χρηστών (Mell & Grance, 2011).

Παράλληλα, ιδιαίτερη σημασία έχουν αποκτήσει οι τεχνολογίες containerization, με κυριότερο εκπρόσωπο το Docker, καθώς και τα συστήματα ορχήστρωσης όπως το Kubernetes. Τα containers επιτρέπουν την εκτέλεση εφαρμογών σε απομονωμένα περιβάλλοντα, διασφαλίζοντας ότι αυτές θα λειτουργούν με τον ίδιο τρόπο ανεξάρτητα από το λειτουργικό σύστημα ή την υποδομή στην οποία εγκαθίστανται. Η χρήση τους διευκολύνει τη μεταφορά εφαρμογών μεταξύ διαφορετικών περιβαλλόντων ανάπτυξης, δοκιμών και παραγωγής, μειώνοντας σημαντικά τα προβλήματα συμβατότητας (Burns et al., 2016).

Η συνεργασία μεταξύ των προγραμματιστών υποστηρίζεται πλέον από προηγμένα συστήματα ελέγχου εκδόσεων (Version Control Systems), με κυρίαρχο παράδειγμα το Git. Μέσω των συστημάτων αυτών είναι δυνατή η ταυτόχρονη ανάπτυξη του ίδιου έργου από πολλές ομάδες, η καταγραφή όλων των αλλαγών στον κώδικα, η επαναφορά προηγούμενων εκδόσεων και η αποτελεσματική διαχείριση διαφορετικών εκδόσεων μιας εφαρμογής. Πλατφόρμες όπως το GitHub και το GitLab έχουν μετατραπεί σε βασικά εργαλεία συνεργασίας και φιλοξενίας έργων

λογισμικού, συμβάλλοντας σημαντικά στην εξάπλωση του ανοικτού λογισμικού και της συνεργατικής ανάπτυξης εφαρμογών (Chacon & Straub, 2014).

Μία ακόμη ιδιαίτερα σημαντική εξέλιξη αποτελεί η υιοθέτηση της φιλοσοφίας DevOps, η οποία επιδιώκει την ενοποίηση των ομάδων ανάπτυξης λογισμικού (Development) και διαχείρισης υποδομών (Operations). Μέσω της αυτοματοποίησης πολλών διαδικασιών, της συνεχούς συνεργασίας και της χρήσης εργαλείων συνεχούς ολοκλήρωσης και συνεχούς διάθεσης (Continuous Integration - Continuous Deployment, CI/CD), οι οργανισμοί επιτυγχάνουν ταχύτερη παράδοση νέων εκδόσεων λογισμικού, καλύτερο έλεγχο ποιότητας και αποτελεσματικότερη αντιμετώπιση προβλημάτων (Humble & Farley, 2010). Η προσέγγιση αυτή έχει πλέον καθιερωθεί ως βασική πρακτική ανάπτυξης μεγάλων πληροφοριακών συστημάτων.

Παράλληλα, ευρεία αποδοχή έχουν γνωρίσει οι ευέλικτες μεθοδολογίες ανάπτυξης (Agile Methodologies), όπως το Scrum και το Kanban. Σε αντίθεση με τα παραδοσιακά μοντέλα ανάπτυξης, οι ευέλικτες προσεγγίσεις βασίζονται στην επαναληπτική ανάπτυξη μικρών λειτουργικών τμημάτων του λογισμικού, στη συνεχή επικοινωνία με τον πελάτη και στην προσαρμογή στις μεταβαλλόμενες απαιτήσεις. Με τον τρόπο αυτό μειώνεται ο κίνδυνος αποτυχίας των έργων και αυξάνεται η ικανοποίηση των χρηστών, καθώς οι βελτιώσεις ενσωματώνονται συνεχώς κατά τη διάρκεια του έργου (Beck et al., 2001).

Τα τελευταία χρόνια αναπτύσσονται επίσης οι πλατφόρμες Low-Code και No-Code, οι οποίες επιτρέπουν τη δημιουργία εφαρμογών με ελάχιστη ή και χωρίς συγγραφή προγραμματιστικού κώδικα. Οι πλατφόρμες αυτές αξιοποιούν οπτικά περιβάλλοντα ανάπτυξης και προκατασκευασμένα στοιχεία, δίνοντας τη δυνατότητα ακόμη και σε χρήστες χωρίς εξειδικευμένες γνώσεις προγραμματισμού να δημιουργούν απλές επιχειρησιακές εφαρμογές. Παρότι δεν μπορούν να αντικαταστήσουν πλήρως την παραδοσιακή ανάπτυξη λογισμικού, συμβάλλουν σημαντικά στην επιτάχυνση της υλοποίησης εφαρμογών μικρής και μεσαίας πολυπλοκότητας (Gartner, 2023).

2.3 Προκλήσεις στη διαδικασία ανάπτυξης λογισμικού

Η ανάπτυξη λογισμικού αποτελεί μία σύνθετη και απαιτητική διαδικασία, η οποία επηρεάζεται από τεχνολογικούς, οργανωτικούς και ανθρώπινους παράγοντες. Παρά τη σημαντική πρόοδο που έχει επιτευχθεί τα τελευταία χρόνια στις μεθοδολογίες ανάπτυξης, στα εργαλεία αυτοματοποίησης και στις υπολογιστικές υποδομές, η υλοποίηση ποιοτικών εφαρμογών εξακολουθεί να συνοδεύεται από σημαντικές προκλήσεις. Οι απαιτήσεις των χρηστών μεταβάλλονται συνεχώς, οι τεχνολογίες εξελίσσονται με ιδιαίτερα γρήγορο ρυθμό και τα πληροφοριακά συστήματα γίνονται ολοένα πιο σύνθετα, γεγονός που αυξάνει σημαντικά τον βαθμό δυσκολίας της ανάπτυξής τους (Pressman & Maxim, 2020).

Μία από τις σημαντικότερες προκλήσεις αφορά τη διαχείριση της πολυπλοκότητας των πληροφοριακών συστημάτων. Οι σύγχρονες εφαρμογές δεν αποτελούνται πλέον από μεμονωμένα προγράμματα, αλλά από ένα σύνολο διασυνδεδεμένων υπηρεσιών, βάσεων δεδομένων, διαδικτυακών εφαρμογών και υποδομών νέφους, οι οποίες πρέπει να συνεργάζονται αποτελεσματικά. Η αυξανόμενη πολυπλοκότητα δυσχεραίνει τόσο τον σχεδιασμό όσο και τη συντήρηση του λογισμικού, ενώ αυξάνει την πιθανότητα εμφάνισης σφαλμάτων και προβλημάτων συμβατότητας μεταξύ διαφορετικών τεχνολογιών (Sommerville, 2011).

Εξίσου σημαντική είναι η πρόκληση της διαχείρισης των συνεχώς μεταβαλλόμενων απαιτήσεων. Κατά τη διάρκεια ενός έργου ανάπτυξης λογισμικού είναι συχνό φαινόμενο οι πελάτες ή οι τελικοί χρήστες να τροποποιούν τις αρχικές τους ανάγκες, είτε λόγω επιχειρησιακών αλλαγών είτε λόγω εμφάνισης νέων τεχνολογικών δυνατοτήτων. Οι αλλαγές αυτές επηρεάζουν τον σχεδιασμό, την υλοποίηση και τις δοκιμές του λογισμικού, αυξάνοντας το κόστος και τον χρόνο ολοκλήρωσης του έργου. Για τον λόγο αυτό, οι σύγχρονες ευέλικτες μεθοδολογίες ανάπτυξης επιδιώκουν τη συνεχή επικοινωνία με τον πελάτη και την επαναληπτική ενσωμάτωση νέων απαιτήσεων (Beck et al., 2001).

Ιδιαίτερη σημασία έχει επίσης η διασφάλιση της ποιότητας του λογισμικού. Η ποιότητα δεν αφορά μόνο τη σωστή λειτουργία μιας εφαρμογής, αλλά και χαρακτηριστικά όπως η αξιοπιστία, η απόδοση, η ασφάλεια, η ευχρηστία και η δυνατότητα συντήρησης. Ένα λογισμικό που λειτουργεί σωστά αλλά παρουσιάζει χαμηλή απόδοση ή δυσκολία στη συντήρηση ενδέχεται να δημιουργήσει σημαντικά προβλήματα στους οργανισμούς που το χρησιμοποιούν. Για τον λόγο

αυτό, η εφαρμογή ολοκληρωμένων διαδικασιών ελέγχου ποιότητας και δοκιμών αποτελεί αναπόσπαστο μέρος της ανάπτυξης λογισμικού (Pressman & Maxim, 2020).

Παράλληλα, σημαντική πρόκληση αποτελεί το τεχνικό χρέος (Technical Debt), το οποίο μπορεί να αυξηθεί όταν κατά την ανάπτυξη ενός συστήματος επιλέγονται γρήγορες ή προσωρινές λύσεις αντί για περισσότερο ολοκληρωμένες και μακροπρόθεσμα βιώσιμες προσεγγίσεις. Η παράλειψη κατάλληλου σχεδιασμού, η ανεπαρκής τεκμηρίωση και η χρήση πρόχειρων λύσεων μπορεί να επιταχύνουν προσωρινά την ανάπτυξη, δημιουργούν όμως σημαντικές δυσκολίες στη συντήρηση και στην επέκταση του λογισμικού στο μέλλον. Η συσσώρευση τεχνικού χρέους μπορεί να αυξήσει το κόστος των μελλοντικών αλλαγών και να δυσχεράνει την ενσωμάτωση νέων λειτουργιών, ιδιαίτερα σε μεγάλα και σύνθετα πληροφοριακά συστήματα (Martin, 2008).

Η κυβερνοασφάλεια (Cybersecurity) αποτελεί επίσης μία από τις μεγαλύτερες προκλήσεις της σύγχρονης ανάπτυξης λογισμικού. Οι κυβερνοεπιθέσεις αυξάνονται συνεχώς σε αριθμό και πολυπλοκότητα, ενώ ακόμη και μικρές αδυναμίες στον προγραμματισμό μπορούν να οδηγήσουν σε σοβαρές παραβιάσεις δεδομένων και οικονομικές απώλειες. Για τον λόγο αυτό, οι απαιτήσεις ασφάλειας πρέπει να λαμβάνονται υπόψη ήδη από τα πρώτα στάδια σχεδιασμού μιας εφαρμογής και να ενσωματώνονται σε ολόκληρο τον κύκλο ζωής ανάπτυξης του λογισμικού, ακολουθώντας την προσέγγιση «Security by Design» (McGraw, 2006).

Ένα ακόμη σημαντικό ζήτημα αφορά τη συντήρηση των παλαιών πληροφοριακών συστημάτων (Legacy Systems). Πολλοί οργανισμοί εξακολουθούν να χρησιμοποιούν εφαρμογές που αναπτύχθηκαν πριν από αρκετές δεκαετίες και βασίζονται σε παρωχημένες τεχνολογίες ή γλώσσες προγραμματισμού. Η αντικατάσταση των συστημάτων αυτών είναι συχνά ιδιαίτερα δαπανηρή ή επικίνδυνη, με αποτέλεσμα να απαιτείται συνεχής συντήρηση και προσαρμογή τους σε νέες επιχειρησιακές ανάγκες και τεχνολογικά περιβάλλοντα (Sommerville, 2011).

Παράλληλα, η ανάπτυξη λογισμικού επηρεάζεται σημαντικά από την έλλειψη εξειδικευμένου ανθρώπινου δυναμικού. Η αυξανόμενη ζήτηση για προγραμματιστές με γνώσεις σε σύγχρονες τεχνολογίες, υπολογιστικό νέφος, κυβερνοασφάλεια και Τεχνητή Νοημοσύνη δυσκολεύει τους οργανισμούς να στελεχώσουν αποτελεσματικά τις ομάδες ανάπτυξης. Η συνεχής εξέλιξη των τεχνολογιών απαιτεί διαρκή εκπαίδευση και αναβάθμιση των δεξιοτήτων των μηχανικών λογισμικού, γεγονός που καθιστά τη δια βίου μάθηση απαραίτητη προϋπόθεση για την επαγγελματική τους εξέλιξη (Pressman & Maxim, 2020).

Κεφάλαιο 3: Η Χρήση της Τεχνητής Νοημοσύνης στην Ανάπτυξη Λογισμικού

3.1 Αυτόματη παραγωγή κώδικα

Η αυτόματη παραγωγή κώδικα (Automatic Code Generation) αποτελεί μία από τις σημαντικότερες εφαρμογές της Τεχνητής Νοημοσύνης στον τομέα της ανάπτυξης λογισμικού. Μέχρι πρόσφατα, η διαδικασία συγγραφής κώδικα βασιζόταν αποκλειστικά στις γνώσεις, την εμπειρία και τις δεξιότητες των προγραμματιστών. Η εμφάνιση, όμως, προηγμένων μοντέλων Τεχνητής Νοημοσύνης, και ιδιαίτερα των Μεγάλων Γλωσσικών Μοντέλων (Large Language Models - LLMs), έχει μεταβάλει σημαντικά τον τρόπο με τον οποίο αναπτύσσεται το λογισμικό. Σήμερα, εργαλεία που βασίζονται στην Τεχνητή Νοημοσύνη μπορούν να δημιουργούν ολόκληρα τμήματα προγραμματιστικού κώδικα, να προτείνουν λύσεις σε προβλήματα, να συμπληρώνουν αυτόματα εντολές και να επιταχύνουν σημαντικά τη διαδικασία ανάπτυξης εφαρμογών (Chen et al., 2021).

Η αυτόματη παραγωγή κώδικα βασίζεται στην εκπαίδευση μοντέλων μηχανικής μάθησης πάνω σε πολύ μεγάλα σύνολα δεδομένων, τα οποία περιλαμβάνουν εκατομμύρια γραμμές προγραμματιστικού κώδικα, τεχνική τεκμηρίωση, αποθετήρια ανοικτού λογισμικού και προγραμματιστικά παραδείγματα. Μέσα από αυτή τη διαδικασία, τα μοντέλα μαθαίνουν τα συντακτικά και σημασιολογικά χαρακτηριστικά διαφορετικών γλωσσών προγραμματισμού, καθώς και τις συνηθέστερες πρακτικές ανάπτυξης λογισμικού. Όταν ο χρήστης διατυπώσει μία οδηγία σε φυσική γλώσσα ή ξεκινήσει να γράφει κώδικα, το μοντέλο προβλέπει τη συνέχεια με βάση τα πρότυπα που έχει μάθει κατά την εκπαίδευσή του και προτείνει αντίστοιχες λύσεις (OpenAI, 2023).

Η εξέλιξη αυτή κατέστη δυνατή χάρη στην ανάπτυξη της αρχιτεκτονικής Transformer, η οποία παρουσιάστηκε από τους Vaswani et al. (2017) και αποτέλεσε τη βάση για τα περισσότερα σύγχρονα γλωσσικά μοντέλα. Σε αντίθεση με προηγούμενες προσεγγίσεις, οι Transformers μπορούν να επεξεργάζονται μεγάλα τμήματα κειμένου ταυτόχρονα, αναγνωρίζοντας σύνθετες σχέσεις μεταξύ των στοιχείων του κώδικα. Έτσι, τα μοντέλα αποκτούν τη δυνατότητα να κατανοούν όχι μόνο τη σύνταξη μιας γλώσσας προγραμματισμού, αλλά και το γενικότερο πλαίσιο στο οποίο αναπτύσσεται μία εφαρμογή.

Τα τελευταία χρόνια έχουν αναπτυχθεί αρκετά εργαλεία που αξιοποιούν τις δυνατότητες αυτές. Ένα από τα πιο γνωστά είναι το GitHub Copilot, το οποίο δημιουργήθηκε από τη GitHub σε συνεργασία με την OpenAI. Το εργαλείο ενσωματώνεται απευθείας στα δημοφιλή περιβάλλοντα ανάπτυξης λογισμικού (Integrated Development Environments - IDEs) και παρέχει προτάσεις κώδικα σε πραγματικό χρόνο, καθώς ο προγραμματιστής εργάζεται. Οι προτάσεις μπορεί να αφορούν μία μόνο γραμμή κώδικα, μία ολόκληρη συνάρτηση ή ακόμη και πλήρεις αλγορίθμους, ανάλογα με το περιεχόμενο του έργου και τις οδηγίες του χρήστη (Chen et al., 2021).

Εκτός από το GitHub Copilot, ιδιαίτερα σημαντική είναι και η συμβολή εφαρμογών γενικής χρήσης, όπως το ChatGPT, το οποίο μπορεί να παράγει κώδικα σε πολλές διαφορετικές γλώσσες προγραμματισμού, να εξηγεί τη λειτουργία αλγορίθμων, να προτείνει βελτιώσεις, να δημιουργεί τεκμηρίωση και να υποστηρίζει την επίλυση προγραμματιστικών προβλημάτων. Αντίστοιχα εργαλεία αποτελούν το Amazon CodeWhisperer, το Tabnine, καθώς και νεότερες πλατφόρμες ανάπτυξης λογισμικού που ενσωματώνουν λειτουργίες δημιουργικής Τεχνητής Νοημοσύνης. Η συνεχής εξέλιξη των εργαλείων αυτών δείχνει ότι η Τεχνητή Νοημοσύνη μετατρέπεται σταδιακά σε έναν ψηφιακό συνεργάτη του προγραμματιστή και όχι απλώς σε ένα βοηθητικό εργαλείο (Tambon et al., 2025).

Ωστόσο, η αυτόματη παραγωγή κώδικα συνοδεύεται και από σημαντικούς περιορισμούς. Παρότι τα σύγχρονα μοντέλα παρουσιάζουν εντυπωσιακές δυνατότητες, δεν κατανοούν πραγματικά τη λειτουργία του λογισμικού με τον ίδιο τρόπο που το αντιλαμβάνεται ένας έμπειρος μηχανικός λογισμικού. Οι απαντήσεις τους βασίζονται σε στατιστικές πιθανότητες και όχι σε πραγματική κατανόηση του προβλήματος. Ως αποτέλεσμα, είναι πιθανό να παραχθεί κώδικας ο οποίος περιέχει λογικά σφάλματα, αναποτελεσματικές λύσεις ή ακόμη και ευπάθειες ασφαλείας, ιδιαίτερα όταν το πρόβλημα είναι πολύπλοκο ή απαιτεί εξειδικευμένη γνώση (Pearce et al., 2022).

3.2 Υποστήριξη προγραμματιστών μέσω εργαλείων AI

Η ενσωμάτωση της Τεχνητής Νοημοσύνης στα εργαλεία ανάπτυξης λογισμικού έχει μεταβάλει ουσιαστικά τον τρόπο με τον οποίο εργάζονται οι προγραμματιστές. Ενώ τα πρώτα εργαλεία ανάπτυξης περιορίζονταν στην επισήμανση συντακτικών λαθών και στην αυτόματη συμπλήρωση λέξεων-κλειδιών, τα σύγχρονα συστήματα Τεχνητής Νοημοσύνης μπορούν να κατανοούν το πλαίσιο ανάπτυξης μιας εφαρμογής, να παρέχουν εξατομικευμένες προτάσεις και να υποστηρίζουν τους προγραμματιστές σε όλα σχεδόν τα στάδια του κύκλου ζωής του λογισμικού. Τα εργαλεία αυτά λειτουργούν ως «έξυπνοι βοηθοί» (AI Assistants), ενισχύοντας την παραγωγικότητα, μειώνοντας τον χρόνο ανάπτυξης και διευκολύνοντας την επίλυση σύνθετων προγραμματιστικών προβλημάτων (Tambon et al., 2025).

Η λειτουργία των περισσότερων εργαλείων βασίζεται σε μεγάλα γλωσσικά μοντέλα (Large Language Models - LLMs), τα οποία έχουν εκπαιδευτεί σε εκτεταμένα αποθετήρια προγραμματιστικού κώδικα, τεχνικής τεκμηρίωσης και επιστημονικού υλικού. Μέσω αυτής της εκπαίδευσης, τα μοντέλα είναι σε θέση να αναγνωρίζουν πρότυπα ανάπτυξης λογισμικού, να προβλέπουν τις πιθανότερες εντολές που επιθυμεί να γράψει ο προγραμματιστής και να προτείνουν ολοκληρωμένες λύσεις. Σε αντίθεση με τα παραδοσιακά εργαλεία αυτόματης συμπλήρωσης, τα οποία βασίζονταν αποκλειστικά στη σύνταξη της γλώσσας προγραμματισμού, τα σύγχρονα εργαλεία λαμβάνουν υπόψη ολόκληρο το πλαίσιο του έργου, επιτυγχάνοντας πολύ μεγαλύτερη ακρίβεια στις προτάσεις τους (OpenAI, 2023).

Ένα από τα σημαντικότερα πλεονεκτήματα των εργαλείων Τεχνητής Νοημοσύνης είναι η δυνατότητα υποστήριξης της διαδικασίας επίλυσης προβλημάτων. Οι προγραμματιστές καλούνται συχνά να αντιμετωπίσουν σύνθετες τεχνικές δυσκολίες, να επιλέξουν κατάλληλους αλγορίθμους ή να ενσωματώσουν νέες τεχνολογίες σε υπάρχουσες εφαρμογές. Τα εργαλεία AI

μπορούν να προτείνουν πιθανές λύσεις, να εξηγούν τον τρόπο λειτουργίας διαφορετικών αλγορίθμων και να παρέχουν παραδείγματα εφαρμογής τους, επιταχύνοντας σημαντικά τη διαδικασία ανάπτυξης. Αντί να αναζητούν πληροφορίες σε πολλαπλές διαδικτυακές πηγές, οι προγραμματιστές μπορούν να λαμβάνουν άμεσα εξατομικευμένες απαντήσεις μέσα από το ίδιο το περιβάλλον ανάπτυξης (Nijkamp et al., 2022).

Ιδιαίτερα σημαντική είναι επίσης η συμβολή των εργαλείων AI στην κατανόηση και συντήρηση υπάρχοντος κώδικα. Σε μεγάλα πληροφοριακά συστήματα, οι ομάδες ανάπτυξης καλούνται συχνά να εργαστούν σε έργα που έχουν δημιουργηθεί από άλλους προγραμματιστές και περιλαμβάνουν χιλιάδες ή ακόμη και εκατομμύρια γραμμές κώδικα. Η κατανόηση της λειτουργίας τους απαιτεί σημαντικό χρόνο και εμπειρία. Τα εργαλεία Τεχνητής Νοημοσύνης μπορούν να αναλύουν τον υπάρχοντα κώδικα, να εξηγούν τη λειτουργία επιμέρους τμημάτων, να δημιουργούν αυτόματα σχόλια (comments) και να συντάσσουν τεκμηρίωση, διευκολύνοντας σημαντικά τη διαδικασία συντήρησης και αναβάθμισης του λογισμικού (Tambon et al., 2025).

Παράλληλα, η Τεχνητή Νοημοσύνη υποστηρίζει τη μετατροπή κώδικα μεταξύ διαφορετικών γλωσσών προγραμματισμού. Η δυνατότητα αυτή είναι ιδιαίτερα χρήσιμη κατά τον εκσυγχρονισμό παλαιών πληροφοριακών συστημάτων, όπου απαιτείται η μεταφορά εφαρμογών από παρωχημένες γλώσσες σε νεότερες τεχνολογίες. Αν και η διαδικασία αυτή εξακολουθεί να απαιτεί ανθρώπινη επίβλεψη, η Τεχνητή Νοημοσύνη μπορεί να επιταχύνει σημαντικά τη μετατροπή, μειώνοντας τον απαιτούμενο χρόνο και το κόστος ανάπτυξης.

Η αξιοποίηση των εργαλείων AI έχει οδηγήσει στην εμφάνιση της έννοιας του AI Pair Programming, σύμφωνα με την οποία ο προγραμματιστής συνεργάζεται με ένα σύστημα Τεχνητής Νοημοσύνης αντί με έναν δεύτερο προγραμματιστή. Το μοντέλο αυτό επιτρέπει τη συνεχή ανταλλαγή ιδεών, την άμεση παραγωγή εναλλακτικών λύσεων και τη γρήγορη επίλυση αποριών κατά τη διάρκεια της ανάπτυξης. Αντί η Τεχνητή Νοημοσύνη να αντικαθιστά τον άνθρωπο, λειτουργεί ως συνεργάτης που ενισχύει τη δημιουργικότητα και την αποτελεσματικότητα της ομάδας ανάπτυξης (Vaithilingam et al., 2022).

3.3 Έλεγχος και εντοπισμός σφαλμάτων

Ο έλεγχος και ο εντοπισμός σφαλμάτων αποτελούν κρίσιμα στάδια του κύκλου ανάπτυξης λογισμικού, καθώς επηρεάζουν άμεσα την ποιότητα, την αξιοπιστία και την ασφάλεια των πληροφοριακών συστημάτων. Ακόμη και ένα μικρό προγραμματιστικό λάθος μπορεί να οδηγήσει σε δυσλειτουργίες, απώλεια δεδομένων, προβλήματα απόδοσης ή σοβαρές ευπάθειες ασφαλείας. Για τον λόγο αυτό, οι διαδικασίες ελέγχου (testing) και αποσφαλμάτωσης (debugging) αποτελούν αναπόσπαστο μέρος κάθε έργου ανάπτυξης λογισμικού. Τα τελευταία χρόνια, η Τεχνητή Νοημοσύνη έχει αναδειχθεί σε ιδιαίτερα σημαντικό εργαλείο για την αυτοματοποίηση και τη βελτίωση αυτών των διαδικασιών, επιτρέποντας τον ταχύτερο εντοπισμό προβλημάτων και τη μείωση του ανθρώπινου φόρτου εργασίας (Pressman & Maxim, 2020).

Παραδοσιακά, ο εντοπισμός σφαλμάτων βασιζόταν κυρίως στη χειροκίνητη εξέταση του κώδικα από τους προγραμματιστές, στη χρήση εργαλείων αποσφαλμάτωσης (debuggers) και στην εκτέλεση δοκιμών με διαφορετικά σενάρια χρήσης. Αν και οι μέθοδοι αυτές εξακολουθούν να χρησιμοποιούνται, απαιτούν σημαντικό χρόνο και εμπειρία, ιδιαίτερα όταν πρόκειται για μεγάλα πληροφοριακά συστήματα με εκατοντάδες χιλιάδες γραμμές κώδικα. Η αυξανόμενη πολυπλοκότητα των εφαρμογών καθιστά δύσκολη την έγκαιρη αναγνώριση όλων των πιθανών προβλημάτων, γεγονός που αυξάνει τον κίνδυνο εμφάνισης σφαλμάτων ακόμη και μετά την παράδοση του λογισμικού στους τελικούς χρήστες (Sommerville, 2011).

Η αξιοποίηση της Τεχνητής Νοημοσύνης έχει μεταβάλει σημαντικά την προσέγγιση αυτή. Μέσω αλγορίθμων μηχανικής μάθησης και μεγάλων γλωσσικών μοντέλων, τα σύγχρονα εργαλεία μπορούν να αναλύουν τον προγραμματιστικό κώδικα, να εντοπίζουν πιθανά σφάλματα και να προτείνουν διορθώσεις πριν ακόμη εκτελεστεί η εφαρμογή. Τα μοντέλα αυτά έχουν εκπαιδευτεί σε εκατομμύρια παραδείγματα κώδικα και είναι σε θέση να αναγνωρίζουν κοινά προγραμματιστικά λάθη, μη ασφαλείς πρακτικές και αποκλίσεις από τα καθιερωμένα πρότυπα ανάπτυξης λογισμικού (Tambon et al., 2025).

Μία από τις σημαντικότερες εφαρμογές της Τεχνητής Νοημοσύνης είναι η στατική ανάλυση κώδικα (Static Code Analysis). Η διαδικασία αυτή πραγματοποιείται χωρίς να απαιτείται η εκτέλεση του προγράμματος και επιτρέπει τον εντοπισμό συντακτικών λαθών, πιθανών

σφαλμάτων λογικής, προβλημάτων απόδοσης και παραβιάσεων των κανόνων προγραμματισμού. Τα εργαλεία AI μπορούν να αναλύουν ολόκληρα έργα λογισμικού, εντοπίζοντας προβληματικά σημεία που ενδέχεται να διαφύγουν της προσοχής των προγραμματιστών. Επιπλέον, είναι σε θέση να προτείνουν εναλλακτικές υλοποιήσεις, συμβάλλοντας στη βελτίωση της ποιότητας και της αναγνωσιμότητας του κώδικα (Pressman & Maxim, 2020).

Παράλληλα, ιδιαίτερη σημασία έχει η δυνατότητα αυτόματου εντοπισμού σφαλμάτων (Bug Detection). Τα σύγχρονα εργαλεία Τεχνητής Νοημοσύνης μπορούν να συγκρίνουν τον παραγόμενο κώδικα με εκατομμύρια προηγούμενα παραδείγματα και να αναγνωρίζουν μοτίβα που συνδέονται με γνωστές κατηγορίες προγραμματιστικών λαθών. Για παράδειγμα, μπορούν να εντοπίσουν πιθανές τιμές μηδενικών αναφορών (null references), ανεπαρκή διαχείριση εξαιρέσεων, προβλήματα συγχρονισμού σε εφαρμογές πολλαπλών νημάτων (multithreading), καθώς και μη ασφαλή διαχείριση δεδομένων εισόδου. Η δυνατότητα αυτή μειώνει σημαντικά τον αριθμό των σφαλμάτων που φτάνουν στο τελικό προϊόν και συμβάλλει στη βελτίωση της αξιοπιστίας του λογισμικού (Pearce et al., 2022).

Η Τεχνητή Νοημοσύνη συμβάλλει επίσης σημαντικά στη διαδικασία του Code Review, δηλαδή της αξιολόγησης του κώδικα πριν από την ενσωμάτωσή του στο τελικό έργο. Παραδοσιακά, ο έλεγχος αυτός πραγματοποιείται από άλλους προγραμματιστές της ομάδας, οι οποίοι εξετάζουν την ποιότητα, τη δομή και την ορθότητα του κώδικα. Τα εργαλεία AI μπορούν πλέον να συμμετέχουν ενεργά στη διαδικασία αυτή, επισημαίνοντας πιθανά προβλήματα, προτείνοντας βελτιώσεις και εντοπίζοντας αποκλίσεις από τα πρότυπα ανάπτυξης που ακολουθεί ο οργανισμός. Με τον τρόπο αυτό μειώνεται ο χρόνος που απαιτείται για τον έλεγχο και αυξάνεται η συνέπεια στην αξιολόγηση του λογισμικού (Tambon et al., 2025).

Ένας ακόμη σημαντικός τομέας εφαρμογής αφορά την αναδόμηση κώδικα (Code Refactoring). Η αναδόμηση δεν μεταβάλλει τη λειτουργικότητα μιας εφαρμογής, αλλά βελτιώνει τη δομή, την αναγνωσιμότητα και τη συντηρησιμότητα του κώδικα. Τα εργαλεία Τεχνητής Νοημοσύνης μπορούν να εντοπίζουν επαναλαμβανόμενο κώδικα, πολύπλοκες συναρτήσεις, περιττές μεταβλητές ή αναποτελεσματικές δομές και να προτείνουν πιο καθαρές και αποδοτικές υλοποιήσεις. Με τον τρόπο αυτό περιορίζεται το τεχνικό χρέος (technical debt) και διευκολύνεται η μελλοντική συντήρηση του λογισμικού (Martin, 2008).

Εξίσου σημαντική είναι η συμβολή της Τεχνητής Νοημοσύνης στη δημιουργία αυτοματοποιημένων δοκιμών (Automated Test Generation). Η ανάπτυξη ολοκληρωμένων δοκιμών αποτελεί μία από τις πιο χρονοβόρες διαδικασίες στην ανάπτυξη λογισμικού. Σήμερα, τα εργαλεία AI μπορούν να δημιουργούν αυτόματα μονάδες ελέγχου (unit tests), σενάρια δοκιμών και δεδομένα εισόδου, καλύπτοντας μεγάλο μέρος των πιθανών περιπτώσεων χρήσης μιας εφαρμογής. Η δυνατότητα αυτή αυξάνει την κάλυψη των δοκιμών, συμβάλλει στον έγκαιρο εντοπισμό σφαλμάτων και μειώνει το κόστος ανάπτυξης (Tambon et al., 2025).

Η Τεχνητή Νοημοσύνη διαδραματίζει επίσης σημαντικό ρόλο στον συνεχή έλεγχο λογισμικού (Continuous Testing), ο οποίος αποτελεί βασικό στοιχείο των πρακτικών DevOps και Continuous Integration/Continuous Deployment (CI/CD). Κάθε φορά που πραγματοποιείται αλλαγή στον κώδικα, τα εργαλεία AI μπορούν να εκτελούν αυτόματα ελέγχους, να αξιολογούν τον βαθμό επικινδυνότητας των αλλαγών και να ενημερώνουν άμεσα τους προγραμματιστές για πιθανά προβλήματα. Η αυτοματοποίηση αυτή επιτρέπει την ταχύτερη ανάπτυξη νέων εκδόσεων λογισμικού, διατηρώντας παράλληλα υψηλό επίπεδο ποιότητας και αξιοπιστίας (Humble & Farley, 2010).

3.4 Αυτοματοποίηση διαδικασιών ανάπτυξης

Η ανάπτυξη λογισμικού αποτελεί μια σύνθετη διαδικασία που περιλαμβάνει μεγάλο αριθμό επαναλαμβανόμενων και χρονοβόρων εργασιών. Εκτός από τη συγγραφή του προγραμματιστικού κώδικα, οι ομάδες ανάπτυξης καλούνται να διαχειριστούν διαδικασίες όπως η ενσωμάτωση νέου κώδικα, η εκτέλεση δοκιμών, η δημιουργία τεκμηρίωσης, η ανάπτυξη νέων εκδόσεων, η παρακολούθηση της λειτουργίας των εφαρμογών και η διαχείριση αλλαγών. Η αυτοματοποίηση των διαδικασιών αυτών έχει εξελιχθεί σε βασικό παράγοντα επιτυχίας των σύγχρονων έργων λογισμικού, καθώς συμβάλλει στη μείωση του χρόνου ανάπτυξης, στον

περιορισμό των ανθρώπινων λαθών και στη βελτίωση της συνολικής ποιότητας του τελικού προϊόντος (Pressman & Maxim, 2020).

Τα τελευταία χρόνια, η Τεχνητή Νοημοσύνη έχει ενισχύσει σημαντικά τις δυνατότητες αυτοματοποίησης, επιτρέποντας όχι μόνο την εκτέλεση προκαθορισμένων ενεργειών αλλά και τη λήψη αποφάσεων με βάση τα δεδομένα που παράγονται κατά τη διάρκεια της ανάπτυξης. Έτσι, η αυτοματοποίηση δεν περιορίζεται πλέον στην εκτέλεση επαναλαμβανόμενων εργασιών, αλλά επεκτείνεται στην ανάλυση πληροφοριών, στην πρόβλεψη προβλημάτων και στη βελτιστοποίηση ολόκληρου του κύκλου ζωής ανάπτυξης λογισμικού (Tambon et al., 2025).

Ένας από τους σημαντικότερους τομείς εφαρμογής της αυτοματοποίησης είναι η Συνεχής Ολοκλήρωση (Continuous Integration - CI). Στο πλαίσιο αυτό, κάθε αλλαγή που πραγματοποιεί ένας προγραμματιστής ενσωματώνεται αυτόματα στον κεντρικό κώδικα του έργου. Με την υποστήριξη εργαλείων αυτοματοποίησης, εκτελούνται άμεσα διαδικασίες μεταγλώττισης, στατικής ανάλυσης και δοκιμών, ώστε να εντοπιστούν πιθανά προβλήματα πριν αυτά επηρεάσουν ολόκληρη την εφαρμογή. Η Τεχνητή Νοημοσύνη μπορεί να αναλύει τα αποτελέσματα των ελέγχων, να αξιολογεί τον βαθμό επικινδυνότητας κάθε αλλαγής και να ενημερώνει τους προγραμματιστές για τα σημεία που απαιτούν ιδιαίτερη προσοχή (Humble & Farley, 2010).

Άμεσα συνδεδεμένη με τη Συνεχή Ολοκλήρωση είναι η Συνεχής Παράδοση και Συνεχής Ανάπτυξη (Continuous Delivery και Continuous Deployment - CD). Μέσω των διαδικασιών αυτών, κάθε νέα έκδοση του λογισμικού μπορεί να εγκαθίσταται αυτόματα σε περιβάλλοντα δοκιμών ή ακόμη και στο περιβάλλον παραγωγής, εφόσον έχουν ολοκληρωθεί επιτυχώς όλοι οι απαραίτητοι έλεγχοι. Η χρήση Τεχνητής Νοημοσύνης επιτρέπει τη συνεχή παρακολούθηση της διαδικασίας ανάπτυξης, την αναγνώριση πιθανών κινδύνων και τη λήψη αποφάσεων σχετικά με το αν μία νέα έκδοση είναι ασφαλής για δημοσίευση. Με τον τρόπο αυτό μειώνονται οι καθυστερήσεις, αυξάνεται η αξιοπιστία των νέων εκδόσεων και περιορίζεται ο κίνδυνος εμφάνισης σοβαρών προβλημάτων μετά την εγκατάστασή τους (Kim et al., 2021).

Η αυτοματοποίηση επεκτείνεται επίσης στη διαχείριση του κύκλου ζωής ανάπτυξης λογισμικού (Software Development Life Cycle - SDLC). Η Τεχνητή Νοημοσύνη μπορεί να υποστηρίζει τον προγραμματισμό έργων, την ανάλυση απαιτήσεων, την κατανομή εργασιών στις ομάδες ανάπτυξης και την παρακολούθηση της προόδου των έργων. Αναλύοντας ιστορικά δεδομένα προηγούμενων έργων, τα συστήματα AI μπορούν να προβλέπουν πιθανές καθυστερήσεις, να εντοπίζουν δραστηριότητες υψηλού κινδύνου και να προτείνουν διορθωτικές ενέργειες πριν εμφανιστούν σημαντικά προβλήματα. Η δυνατότητα αυτή συμβάλλει στη βελτίωση της διαχείρισης έργων και στην αποτελεσματικότερη αξιοποίηση των διαθέσιμων πόρων (Pressman & Maxim, 2020).

Ιδιαίτερα σημαντική είναι και η συμβολή της Τεχνητής Νοημοσύνης στην αυτοματοποιημένη δημιουργία τεκμηρίωσης. Η τεκμηρίωση αποτελεί απαραίτητο στοιχείο κάθε πληροφοριακού συστήματος, καθώς διευκολύνει τόσο την ανάπτυξη όσο και τη μελλοντική συντήρησή του. Παρ' όλα αυτά, η σύνταξή της αποτελεί συχνά μία διαδικασία που παραμελείται λόγω περιορισμών χρόνου. Τα σύγχρονα εργαλεία Τεχνητής Νοημοσύνης μπορούν να δημιουργούν αυτόματα τεχνικά έγγραφα, σχόλια μέσα στον κώδικα, οδηγίες εγκατάστασης, περιγραφές συναρτήσεων και εγχειρίδια χρήσης, μειώνοντας σημαντικά τον απαιτούμενο χρόνο και αυξάνοντας την πληρότητα της τεκμηρίωσης (OpenAI, 2023).

Παράλληλα, η αυτοματοποίηση αξιοποιείται στη διαχείριση απαιτήσεων λογισμικού. Τα εργαλεία AI μπορούν να αναλύουν έγγραφα προδιαγραφών, να εντοπίζουν ασάφειες ή αντιφάσεις στις απαιτήσεις και να προτείνουν βελτιώσεις πριν ξεκινήσει η ανάπτυξη του λογισμικού. Επιπλέον, έχουν τη δυνατότητα να συγκρίνουν νέες απαιτήσεις με ήδη υπάρχουσες λειτουργίες του συστήματος και να εκτιμούν τον πιθανό αντίκτυπο που θα έχουν οι αλλαγές αυτές στην αρχιτεκτονική της εφαρμογής. Με τον τρόπο αυτό μειώνεται ο κίνδυνος λανθασμένου σχεδιασμού και διευκολύνεται η λήψη τεκμηριωμένων αποφάσεων από τις ομάδες ανάπτυξης.

Ένας ακόμη τομέας όπου η Τεχνητή Νοημοσύνη έχει αποκτήσει ιδιαίτερη σημασία είναι η παρακολούθηση και διαχείριση λειτουργίας των εφαρμογών (Application Monitoring). Μετά την ανάπτυξη ενός πληροφοριακού συστήματος, τα εργαλεία AI μπορούν να συλλέγουν συνεχώς δεδομένα σχετικά με την απόδοση, τη χρήση πόρων και τη συμπεριφορά των χρηστών. Μέσω τεχνικών μηχανικής μάθησης είναι δυνατόν να εντοπίζονται ανωμαλίες, να προβλέπονται πιθανές αστοχίες του συστήματος και να ενεργοποιούνται αυτόματα διαδικασίες αποκατάστασης πριν οι

τελικοί χρήστες αντιληφθούν κάποιο πρόβλημα. Η πρακτική αυτή είναι γνωστή ως AIOps (Artificial Intelligence for IT Operations) και χρησιμοποιείται ολοένα και περισσότερο από μεγάλους οργανισμούς για τη βελτίωση της διαθεσιμότητας και της αξιοπιστίας των πληροφοριακών συστημάτων (IBM, 2023).

Κεφάλαιο 4: Πλεονεκτήματα και Μειονεκτήματα της Χρήσης AI

4.1 Αύξηση παραγωγικότητας και ταχύτητας ανάπτυξης

Η αύξηση της παραγωγικότητας αποτελεί ένα από τα σημαντικότερα πλεονεκτήματα που συνδέονται με την αξιοποίηση της Τεχνητής Νοημοσύνης στην ανάπτυξη λογισμικού. Η δημιουργία μιας εφαρμογής περιλαμβάνει πολλές εργασίες που απαιτούν χρόνο, όπως η συγγραφή και επεξεργασία κώδικα, η αναζήτηση τεχνικών λύσεων, η δημιουργία δοκιμών, η τεκμηρίωση και η διόρθωση σφαλμάτων. Τα σύγχρονα εργαλεία Τεχνητής Νοημοσύνης μπορούν να αναλάβουν ή να υποστηρίξουν μέρος αυτών των εργασιών, επιτρέποντας στον προγραμματιστή να επικεντρωθεί περισσότερο στον σχεδιασμό, στην επίλυση σύνθετων προβλημάτων και στη λήψη αποφάσεων που απαιτούν ανθρώπινη κρίση (Peng et al., 2023).

Ιδιαίτερα σημαντική είναι η δυνατότητα των εργαλείων παραγωγικής Τεχνητής Νοημοσύνης να δημιουργούν κώδικα με βάση οδηγίες που δίνονται σε φυσική γλώσσα. Αντί ο προγραμματιστής να γράφει κάθε εντολή από την αρχή, μπορεί να περιγράψει τη λειτουργία που επιθυμεί και το σύστημα να δημιουργήσει μία αρχική υλοποίηση. Παράλληλα, εργαλεία όπως το GitHub Copilot μπορούν να προτείνουν τη συνέχεια του κώδικα καθώς αυτός γράφεται, μειώνοντας τον χρόνο που απαιτείται για συνηθισμένες και επαναλαμβανόμενες εργασίες. Η δυνατότητα αυτή είναι ιδιαίτερα χρήσιμη σε περιπτώσεις όπου απαιτείται η δημιουργία τυποποιημένων συναρτήσεων, βασικών δομών δεδομένων ή επαναλαμβανόμενων τμημάτων κώδικα (Zhang et al., 2023).

Τα αποτελέσματα εμπειρικών ερευνών δείχνουν ότι, υπό συγκεκριμένες συνθήκες, η χρήση τέτοιων εργαλείων μπορεί να επιφέρει σημαντική εξοικονόμηση χρόνου. Στο ελεγχόμενο πείραμα των Peng et al. (2023), προγραμματιστές που είχαν πρόσβαση στο GitHub Copilot κλήθηκαν να δημιουργήσουν έναν HTTP server σε JavaScript και ολοκλήρωσαν την εργασία κατά περίπου 55,8% ταχύτερα από την ομάδα που δεν χρησιμοποίησε το εργαλείο. Το συγκεκριμένο εύρημα αποτελεί σημαντική ένδειξη της δυνατότητας των AI coding assistants να αυξήσουν την ταχύτητα εκτέλεσης συγκεκριμένων προγραμματιστικών εργασιών, χωρίς όμως να σημαίνει ότι το ίδιο ποσοστό βελτίωσης μπορεί να γενικευθεί σε κάθε έργο ανάπτυξης λογισμικού.

Η αύξηση της παραγωγικότητας δεν σχετίζεται αποκλειστικά με την ταχύτερη συγγραφή κώδικα. Ένα σημαντικό μέρος του χρόνου ενός προγραμματιστή αφιερώνεται στην αναζήτηση πληροφοριών, στη μελέτη τεχνικής τεκμηρίωσης και στην προσπάθεια κατανόησης τεχνολογιών με τις οποίες δεν είναι εξοικειωμένος. Τα εργαλεία AI μπορούν να παρέχουν άμεσες επεξηγήσεις για συναρτήσεις, βιβλιοθήκες και προγραμματιστικές έννοιες, να προτείνουν παραδείγματα και να παρουσιάζουν εναλλακτικούς τρόπους αντιμετώπισης ενός προβλήματος. Επομένως, λειτουργούν όχι μόνο ως εργαλεία παραγωγής κώδικα αλλά και ως μηχανισμοί άμεσης τεχνικής υποστήριξης του προγραμματιστή.

Παράλληλα, η παραγωγικότητα μπορεί να ενισχυθεί μέσω της μείωσης του γνωστικού φόρτου που προκαλούν οι επαναλαμβανόμενες εργασίες. Όταν ένα σύστημα αναλαμβάνει τη δημιουργία τυποποιημένων τμημάτων κώδικα ή προτείνει πιθανές λύσεις, ο προγραμματιστής μπορεί να αφιερώσει περισσότερο χρόνο σε εργασίες που απαιτούν δημιουργικότητα και κριτική σκέψη. Έρευνα σχετικά με τη χρήση του GitHub Copilot έδειξε ότι οι χρήστες συχνά αντιλαμβάνονται τα εργαλεία αυτά ως έναν τρόπο περιορισμού των επαναλαμβανόμενων εργασιών και του γνωστικού φόρτου, γεγονός που μπορεί να επηρεάσει θετικά όχι μόνο την ταχύτητα αλλά και την εμπειρία της εργασίας τους.

Η Τεχνητή Νοημοσύνη μπορεί επίσης να επιταχύνει τη διαδικασία ανάπτυξης σε επίπεδο ολόκληρης ομάδας. Η αυτόματη δημιουργία τεκμηρίωσης, η υποστήριξη κατά τη δημιουργία δοκιμών και η παροχή προτάσεων κατά την αναθεώρηση κώδικα μπορούν να περιορίσουν τον χρόνο που αφιερώνεται σε βοηθητικές δραστηριότητες. Με αυτόν τον τρόπο, η ομάδα μπορεί να ολοκληρώνει ταχύτερα μικρότερες αλλαγές και να διαθέτει περισσότερο χρόνο για την ανάπτυξη

νέων λειτουργιών. Η επίδραση αυτή αποκτά μεγαλύτερη σημασία σε περιβάλλοντα Agile και DevOps, όπου οι ομάδες επιδιώκουν συχνές αλλαγές και μικρούς κύκλους ανάπτυξης.

Ωστόσο, η σχέση μεταξύ χρήσης Τεχνητής Νοημοσύνης και αυξημένης παραγωγικότητας δεν είναι απόλυτη. Η ταχύτητα με την οποία παράγεται κώδικας δεν ταυτίζεται απαραίτητα με την ταχύτητα ολοκλήρωσης ενός ποιοτικού έργου λογισμικού. Ο κώδικας που δημιουργείται από ένα σύστημα AI πρέπει να διαβαστεί, να κατανοηθεί, να ελεγχθεί και να δοκιμαστεί πριν ενσωματωθεί στην τελική εφαρμογή. Εάν περιέχει λάθη ή δεν ανταποκρίνεται πλήρως στις απαιτήσεις του έργου, ο χρόνος που εξοικονομήθηκε κατά τη δημιουργία του μπορεί στη συνέχεια να δαπανηθεί για την επανεξέταση και τη διόρθωσή του.

Το σημείο αυτό επιβεβαιώνεται και από νεότερα εμπειρικά δεδομένα, τα οποία δείχνουν ότι η επίδραση της AI στην παραγωγικότητα εξαρτάται σε μεγάλο βαθμό από το είδος της εργασίας και το επίπεδο εμπειρίας του προγραμματιστή. Σε τυχαιοποιημένη μελέτη της METR με 16 έμπειρους προγραμματιστές ανοικτού λογισμικού και 246 πραγματικές εργασίες σε ώριμα έργα με τα οποία ήταν ήδη ιδιαίτερα εξοικειωμένοι, η χρήση εργαλείων AI στις αρχές του 2025 συνδέθηκε με αύξηση του χρόνου ολοκλήρωσης κατά 19%. Οι ίδιοι οι συμμετέχοντες, μάλιστα, θεωρούσαν ότι η AI τούς έκανε ταχύτερους, γεγονός που δείχνει ότι η υποκειμενική αντίληψη παραγωγικότητας δεν συμπίπτει πάντοτε με την πραγματικά μετρήσιμη εξοικονόμηση χρόνου (Becker et al., 2025).

Το εύρημα αυτό δεν αναιρεί τα προηγούμενα αποτελέσματα, αλλά αναδεικνύει ότι δεν είναι επιστημονικά ορθό να υποστηρίζεται πως η Τεχνητή Νοημοσύνη αυξάνει αυτομάτως την παραγωγικότητα κάθε προγραμματιστή. Στην πράξη, το αποτέλεσμα επηρεάζεται από παράγοντες όπως η πολυπλοκότητα του έργου, η εμπειρία του χρήστη, η εξοικείωσή του με τον υπάρχοντα κώδικα, η ποιότητα των προτάσεων του μοντέλου και ο χρόνος που απαιτείται για την επαλήθευσή τους. Αξίζει επίσης να σημειωθεί ότι η ίδια η METR ανέφερε το 2026 ότι τα νεότερα εργαλεία είναι πιθανό να προσφέρουν μεγαλύτερη επιτάχυνση σε σχέση με εκείνα των αρχών του 2025, αλλά τα νεότερα δεδομένα της δεν επιτρέπουν ακόμη ασφαλή ποσοτική εκτίμηση λόγω μεθοδολογικών περιορισμών (METR, 2026).

Ένας ακόμη παράγοντας που πρέπει να εξεταστεί είναι ο χρόνος που απαιτείται για τη διατύπωση κατάλληλων οδηγιών προς το σύστημα. Η αποτελεσματικότητα ενός AI εργαλείου εξαρτάται σε σημαντικό βαθμό από το κατά πόσο ο προγραμματιστής μπορεί να περιγράψει με σαφήνεια το πρόβλημα και να αξιολογήσει την απάντηση που λαμβάνει. Όταν οι αρχικές προτάσεις είναι λανθασμένες ή ελλιπείς, μπορεί να απαιτούνται επαναλαμβανόμενες οδηγίες, τροποποιήσεις και διορθώσεις. Σε ορισμένες περιπτώσεις, επομένως, η διαδικασία αυτή μπορεί να γίνει εξίσου χρονοβόρα με τη χειροκίνητη επίλυση του προβλήματος.

4.2 Επίδραση της Τεχνητής Νοημοσύνης στην ποιότητα του λογισμικού

Η ποιότητα αποτελεί βασικό ζητούμενο στη διαδικασία ανάπτυξης λογισμικού, καθώς δεν περιορίζεται μόνο στην ορθή εκτέλεση των λειτουργιών μιας εφαρμογής, αλλά συνδέεται και με χαρακτηριστικά όπως η αξιοπιστία, η ασφάλεια, η αποδοτικότητα, η συντηρησιμότητα και η ευκολία κατανόησης του κώδικα. Η ενσωμάτωση εργαλείων Τεχνητής Νοημοσύνης στη δημιουργία λογισμικού έχει δημιουργήσει νέες δυνατότητες για την υποστήριξη αυτών των χαρακτηριστικών, χωρίς όμως να εξασφαλίζει από μόνη της καλύτερη ποιότητα. Αντίθετα, ανάλογα με τον τρόπο χρήσης των εργαλείων και τον βαθμό ανθρώπινου ελέγχου, η Τεχνητή Νοημοσύνη μπορεί τόσο να συμβάλει στη βελτίωση όσο και να οδηγήσει σε υποβάθμιση της ποιότητας του παραγόμενου λογισμικού (Pearce et al., 2022).

Αρχικά, μία θετική επίδραση της Τεχνητής Νοημοσύνης αφορά τη δυνατότητα υποστήριξης του προγραμματιστή κατά τη συγγραφή του κώδικα. Τα σύγχρονα εργαλεία παραγωγής κώδικα μπορούν να προτείνουν ολοκληρωμένες συναρτήσεις, να συμπληρώνουν τμήματα κώδικα και να παρουσιάζουν εναλλακτικές λύσεις σε ένα προγραμματιστικό πρόβλημα. Η δυνατότητα αυτή μπορεί να περιορίσει ορισμένα συνηθισμένα λάθη και να βοηθήσει τον προγραμματιστή να ακολουθεί περισσότερο συνεπείς πρακτικές ανάπτυξης. Παράλληλα, τα εργαλεία αυτά μπορούν να χρησιμοποιηθούν για την αναδιαμόρφωση υπάρχοντος κώδικα, ώστε αυτός να γίνεται περισσότερο κατανοητός και ευκολότερος στη μελλοντική συντήρησή του (Tambon et al., 2025).

Σημαντική είναι επίσης η συμβολή της Τεχνητής Νοημοσύνης στον εντοπισμό σφαλμάτων και στην υποστήριξη των διαδικασιών ελέγχου. Όπως αναφέρθηκε στην προηγούμενη ενότητα,

τα συστήματα AI μπορούν να εξετάζουν τον κώδικα, να εντοπίζουν πιθανά προβλήματα και να προτείνουν διορθώσεις. Στην παρούσα ενότητα, ωστόσο, το ενδιαφέρον δεν βρίσκεται στην ίδια τη διαδικασία του ελέγχου, αλλά στην επίδραση που μπορεί να έχει στην τελική ποιότητα του λογισμικού. Ο έγκαιρος εντοπισμός ενός σφάλματος πριν από την ενσωμάτωση του κώδικα στο τελικό σύστημα μπορεί να περιορίσει την πιθανότητα εμφάνισης δυσλειτουργιών κατά τη χρήση της εφαρμογής και να συμβάλει στην αύξηση της αξιοπιστίας της (Pressman & Maxim, 2020).

Αντίστοιχα, η Τεχνητή Νοημοσύνη μπορεί να υποστηρίξει τη δημιουργία δοκιμών λογισμικού. Η παραγωγή κατάλληλων περιπτώσεων ελέγχου αποτελεί σημαντικό στοιχείο της διασφάλισης ποιότητας, καθώς επιτρέπει να εξεταστεί εάν μια εφαρμογή ανταποκρίνεται στις απαιτήσεις για τις οποίες σχεδιάστηκε. Η αυτοματοποιημένη δημιουργία δοκιμών μπορεί να βοηθήσει τον προγραμματιστή να εξετάσει περισσότερες περιπτώσεις λειτουργίας και να εντοπίσει προβλήματα τα οποία διαφορετικά ενδέχεται να παρέμεναν απαρατήρητα. Ωστόσο, η ποιότητα των δοκιμών που παράγονται από ένα σύστημα Τεχνητής Νοημοσύνης πρέπει επίσης να αξιολογείται, καθώς ένας μεγάλος αριθμός αυτοματοποιημένων δοκιμών δεν σημαίνει απαραίτητα και αποτελεσματικότερο έλεγχο του λογισμικού (Tambon et al., 2025).

Μία ακόμη θετική διάσταση αφορά τη συντηρησιμότητα του κώδικα. Στην πράξη, μεγάλο μέρος του λογισμικού συνεχίζει να τροποποιείται για μεγάλο χρονικό διάστημα μετά την αρχική του ανάπτυξη. Επομένως, η ποιότητά του εξαρτάται και από το κατά πόσο ένας προγραμματιστής μπορεί να κατανοήσει, να διορθώσει και να επεκτείνει τον υπάρχοντα κώδικα. Εργαλεία AI μπορούν να εξηγούν σύνθετα τμήματα κώδικα, να δημιουργούν σχόλια και τεκμηρίωση και να προτείνουν απλούστερες δομές. Όταν οι προτάσεις αυτές είναι κατάλληλες και ελέγχονται από τον προγραμματιστή, μπορούν να συμβάλουν στη δημιουργία περισσότερο κατανοητού και διαχειρίσιμου λογισμικού (Pressman & Maxim, 2020).

Η παραπάνω θετική εικόνα, όμως, αποτελεί μόνο τη μία πλευρά του ζητήματος. Η χρήση της Τεχνητής Νοημοσύνης μπορεί επίσης να οδηγήσει σε υποβάθμιση της ποιότητας του λογισμικού, στοιχείο που πρέπει να λαμβάνεται σοβαρά υπόψη κατά την αξιολόγηση των συγκεκριμένων τεχνολογιών. Το γεγονός ότι ένα σύστημα μπορεί να παράγει κώδικα που είναι συντακτικά σωστός δεν σημαίνει ότι ο κώδικας αυτός αποτελεί και την κατάλληλη λύση για το συγκεκριμένο πρόβλημα. Μπορεί να εκτελείται χωρίς εμφανή σφάλματα, αλλά να περιλαμβάνει λανθασμένη λογική, περιττή πολυπλοκότητα, αναποτελεσματικές επιλογές ή συμπεριφορές που εμφανίζονται μόνο κάτω από συγκεκριμένες συνθήκες (Pearce et al., 2022).

Το πρόβλημα αυτό συνδέεται με τον τρόπο λειτουργίας των μεγάλων γλωσσικών μοντέλων. Τα μοντέλα δεν αξιολογούν τον κώδικα με τον ίδιο τρόπο που ένας μηχανικός λογισμικού κατανοεί συνολικά τις απαιτήσεις και τους στόχους ενός συστήματος. Παράγουν αποτελέσματα με βάση πρότυπα που έχουν προκύψει από τα δεδομένα εκπαίδευσής τους και από το περιεχόμενο που τους παρέχει ο χρήστης. Συνεπώς, μία απάντηση μπορεί να φαίνεται πειστική και τεχνικά ολοκληρωμένη χωρίς να είναι απαραίτητα ορθή ή κατάλληλη για την εφαρμογή στην οποία πρόκειται να χρησιμοποιηθεί. Αυτό δημιουργεί ιδιαίτερο κίνδυνο όταν ο χρήστης δεν διαθέτει τις απαιτούμενες γνώσεις για να αναγνωρίσει το πρόβλημα και αποδέχεται την προτεινόμενη λύση χωρίς επαρκή έλεγχο (Chen et al., 2021).

Ειδικότερα, προβληματισμό προκαλεί η ποιότητα του κώδικα ως προς την ασφάλεια. Η μελέτη των Pearce et al. (2022), η οποία εξέτασε προτάσεις κώδικα που δημιουργήθηκαν με τη βοήθεια του GitHub Copilot σε διαφορετικά σενάρια υψηλού κινδύνου, έδειξε ότι ένα σημαντικό μέρος των παραγόμενων προγραμμάτων παρουσίαζε αδυναμίες ασφάλειας. Το εύρημα αυτό υπογραμμίζει ότι η λειτουργικότητα δεν πρέπει να θεωρείται το μοναδικό κριτήριο ποιότητας. Ένα πρόγραμμα μπορεί να εκτελεί σωστά τη λειτουργία που του ζητήθηκε και ταυτόχρονα να περιλαμβάνει μία ευπάθεια η οποία θα μπορούσε να αξιοποιηθεί κακόβουλα (Pearce et al., 2022).

Ένα επιπλέον ζήτημα είναι η πιθανότητα δημιουργίας τεχνικού χρέους. Η ευκολία με την οποία παράγονται μεγάλες ποσότητες κώδικα μπορεί να οδηγήσει στην ενσωμάτωση περισσότερων λύσεων χωρίς προηγουμένως να έχει εξεταστεί επαρκώς η συμβατότητά τους με τη συνολική αρχιτεκτονική του συστήματος. Εάν ο προγραμματιστής αποδέχεται διαδοχικά προτάσεις AI χωρίς να εξετάζει τη δομή και τη μακροπρόθεσμη συντηρησιμότητά τους, μπορεί να δημιουργηθεί ένας κώδικας που λειτουργεί βραχυπρόθεσμα αλλά γίνεται σταδιακά δυσκολότερος στην κατανόηση, στη διόρθωση και στην επέκταση. Με αυτόν τον τρόπο, η ταχύτερη παραγωγή κώδικα μπορεί να μετατραπεί σε αυξημένο κόστος συντήρησης στο μέλλον.

Η ποιότητα μπορεί να επηρεαστεί και από την υπερβολική εμπιστοσύνη στις προτάσεις της Τεχνητής Νοημοσύνης. Η ταχύτητα με την οποία παράγεται μία ολοκληρωμένη και φαινομενικά πειστική απάντηση μπορεί να δημιουργήσει την εντύπωση ότι η λύση έχει ήδη ελεγχθεί. Στην πραγματικότητα, ο προγραμματιστής παραμένει υπεύθυνος για την κατανόηση και την επαλήθευση του κώδικα πριν από την ενσωμάτωσή του. Μελέτες για την αλληλεπίδραση των προγραμματιστών με AI pair programmers έχουν δείξει ότι η αξιολόγηση των προτάσεων αποτελεί ουσιαστικό μέρος της χρήσης τους και ότι η χρησιμότητα ενός εργαλείου δεν εξαρτάται μόνο από την ικανότητά του να παράγει κώδικα αλλά και από το κατά πόσο ο χρήστης μπορεί να κατανοήσει και να ελέγξει το αποτέλεσμα (Vaithilingam et al., 2022).

Επομένως, είναι σημαντικό να γίνει διάκριση μεταξύ της ποιότητας του κώδικα που παράγεται από την AI και της ποιότητας του τελικού λογισμικού που αναπτύσσεται με τη βοήθειά της. Τα δύο μεγέθη δεν είναι ταυτόσημα. Μία πρόταση που δεν είναι αρχικά άριστη μπορεί να αποτελέσει χρήσιμη βάση εφόσον ένας έμπειρος προγραμματιστής την ελέγξει, τη διορθώσει και την προσαρμόσει στις απαιτήσεις του έργου. Αντίθετα, ακόμη και μία φαινομενικά καλή πρόταση μπορεί να επηρεάσει αρνητικά το τελικό σύστημα εάν ενσωματωθεί χωρίς να εξεταστούν οι επιπτώσεις της στην ασφάλεια, στην απόδοση, στην αρχιτεκτονική και στη συντηρησιμότητα του λογισμικού.

Κατά συνέπεια, η Τεχνητή Νοημοσύνη δεν πρέπει να αντιμετωπίζεται ως ένας μηχανισμός που οδηγεί αυτομάτως σε βελτίωση της ποιότητας. Η πραγματική της αξία εξαρτάται από τον τρόπο με τον οποίο εντάσσεται στη διαδικασία ανάπτυξης, από την ποιότητα των δεδομένων και των μοντέλων που χρησιμοποιούνται και, κυρίως, από την ανθρώπινη επίβλεψη. Οι διαδικασίες code review, testing, στατικής ανάλυσης και ελέγχου ασφάλειας εξακολουθούν να είναι απαραίτητες ακόμη και όταν ο κώδικας έχει δημιουργηθεί ή ελεγχθεί με τη βοήθεια Τεχνητής Νοημοσύνης (Pearce et al., 2022).

4.3 Ζητήματα αξιοπιστίας και ασφάλειας

Η αξιοπιστία και η ασφάλεια αποτελούν δύο από τα σημαντικότερα ζητήματα που προκύπτουν από την ενσωμάτωση της Τεχνητής Νοημοσύνης στη διαδικασία ανάπτυξης λογισμικού. Παρότι τα σύγχρονα εργαλεία AI μπορούν να παράγουν κώδικα σε πολύ μικρό χρονικό διάστημα και να υποστηρίζουν τους προγραμματιστές στην επίλυση σύνθετων προβλημάτων, οι απαντήσεις τους δεν είναι πάντοτε ορθές, ασφαλείς ή κατάλληλες για το συγκεκριμένο περιβάλλον στο οποίο πρόκειται να χρησιμοποιηθούν. Για τον λόγο αυτό, η αξιοποίηση κώδικα που παράγεται από Τεχνητή Νοημοσύνη απαιτεί συστηματικό έλεγχο και δεν μπορεί να θεωρείται αυτομάτως αξιόπιστη μόνο και μόνο επειδή το αποτέλεσμα φαίνεται τεχνικά ολοκληρωμένο (Chen et al., 2021).

Ένα βασικό πρόβλημα σχετίζεται με τη δυνατότητα των μεγάλων γλωσσικών μοντέλων να παράγουν λανθασμένες απαντήσεις με ιδιαίτερα πειστικό τρόπο. Τα μοντέλα αυτά δημιουργούν κώδικα με βάση πρότυπα που έχουν μάθει κατά την εκπαίδευσή τους και το περιεχόμενο που παρέχεται από τον χρήστη, χωρίς αυτό να σημαίνει ότι μπορούν να εγγυηθούν την ορθότητα κάθε παραγόμενης λύσης. Έτσι, ένα απόσπασμα κώδικα μπορεί να είναι συντακτικά σωστό και να φαίνεται λογικό, αλλά να μην ανταποκρίνεται πλήρως στις πραγματικές απαιτήσεις του συστήματος ή να αποτυγχάνει σε συγκεκριμένες περιπτώσεις χρήσης. Το πρόβλημα γίνεται περισσότερο σοβαρό όταν ο χρήστης εμπιστεύεται την απάντηση χωρίς να πραγματοποιεί ανεξάρτητο έλεγχο και δοκιμές (Chen et al., 2021).

Η αξιοπιστία των απαντήσεων επηρεάζεται επίσης από τον τρόπο με τον οποίο διατυπώνεται το αίτημα προς το σύστημα. Διαφορετικές οδηγίες μπορεί να οδηγήσουν σε διαφορετικές λύσεις για το ίδιο προγραμματιστικό πρόβλημα, ενώ μία ασαφής ή ελλιπής περιγραφή των απαιτήσεων αυξάνει την πιθανότητα παραγωγής ακατάλληλου κώδικα. Το γεγονός αυτό αποκτά ιδιαίτερη σημασία στην ανάπτυξη πραγματικών εφαρμογών, όπου οι απαιτήσεις είναι συχνά σύνθετες και συνδέονται μεταξύ τους. Συνεπώς, η αποτελεσματική χρήση ενός εργαλείου AI προϋποθέτει ότι ο προγραμματιστής είναι σε θέση τόσο να περιγράψει με σαφήνεια το πρόβλημα όσο και να αξιολογήσει κριτικά την προτεινόμενη λύση (Vaithilingam et al., 2022).

Εκτός από την ορθότητα του παραγόμενου κώδικα, ιδιαίτερο προβληματισμό δημιουργεί η ασφάλειά του. Ένα σύστημα Τεχνητής Νοημοσύνης μπορεί να προτείνει κώδικα που επιτελεί τη

ζητούμενη λειτουργία αλλά ταυτόχρονα περιλαμβάνει ευπάθειες. Τέτοιες αδυναμίες μπορεί να αφορούν, μεταξύ άλλων, ανεπαρκή έλεγχο των δεδομένων εισόδου, μη ασφαλή διαχείριση ευαίσθητων πληροφοριών ή λανθασμένη εφαρμογή μηχανισμών προστασίας. Εάν τέτοιες προτάσεις ενσωματωθούν σε πραγματικές εφαρμογές χωρίς προηγούμενο έλεγχο, μπορούν να δημιουργήσουν λογισμικό ευάλωτο σε επιθέσεις και να αυξήσουν τον κίνδυνο παραβίασης ενός πληροφοριακού συστήματος (Pearce et al., 2022).

Χαρακτηριστική είναι η έρευνα των Pearce et al. (2022), οι οποίοι αξιολόγησαν την ασφάλεια κώδικα που παρήχθη με τη βοήθεια του GitHub Copilot σε διαφορετικά σενάρια. Τα αποτελέσματα έδειξαν ότι ένα σημαντικό μέρος των προτάσεων που εξετάστηκαν περιείχε αδυναμίες σχετικές με την ασφάλεια. Το εύρημα αυτό είναι ιδιαίτερα σημαντικό, καθώς δείχνει ότι η ικανότητα ενός συστήματος να δημιουργεί λειτουργικό κώδικα δεν συνεπάγεται ότι ο συγκεκριμένος κώδικας ακολουθεί και τις απαιτούμενες πρακτικές ασφαλούς ανάπτυξης. Επομένως, οι προτάσεις των εργαλείων AI πρέπει να αντιμετωπίζονται ως προτεινόμενες λύσεις που χρειάζονται αξιολόγηση και όχι ως έτοιμος και ελεγμένος κώδικας (Pearce et al., 2022).

Ένα ακόμη ζήτημα αφορά την πιθανότητα αναπαραγωγής παρωχημένων ή μη ασφαλών προγραμματιστικών πρακτικών. Τα μοντέλα παραγωγής κώδικα εκπαιδεύονται σε πολύ μεγάλες συλλογές δεδομένων, στις οποίες μπορεί να περιλαμβάνονται παραδείγματα διαφορετικής ποιότητας. Επομένως, το γεγονός ότι μία συγκεκριμένη προγραμματιστική λύση εμφανίζεται στα δεδομένα εκπαίδευσης δεν σημαίνει ότι αποτελεί και τη βέλτιστη ή ασφαλέστερη πρακτική. Η Bender et al. (2021) επισημαίνουν γενικότερα ότι τα μεγάλα γλωσσικά μοντέλα μπορούν να αναπαράγουν χαρακτηριστικά και προβλήματα που υπάρχουν στα δεδομένα στα οποία βασίζονται. Στο πλαίσιο της ανάπτυξης λογισμικού, αυτό ενισχύει την ανάγκη ελέγχου του παραγόμενου αποτελέσματος πριν από την πρακτική αξιοποίησή του.

Ιδιαίτερη προσοχή απαιτείται και ως προς την προστασία των δεδομένων και του ιδιόκτητου κώδικα. Κατά τη χρήση εξωτερικών υπηρεσιών Τεχνητής Νοημοσύνης, ένας προγραμματιστής μπορεί να εισαγάγει τμήματα πηγαίου κώδικα, τεχνικές πληροφορίες, αρχεία ρυθμίσεων ή άλλα στοιχεία ενός έργου προκειμένου να ζητήσει βοήθεια από το σύστημα. Εάν οι πληροφορίες αυτές περιλαμβάνουν εμπιστευτικά επιχειρησιακά δεδομένα, προσωπικά δεδομένα, διαπιστευτήρια πρόσβασης ή άλλο ευαίσθητο περιεχόμενο, η μη ελεγχόμενη κοινοποίησή τους μπορεί να δημιουργήσει κινδύνους για τον οργανισμό. Για τον λόγο αυτό, η χρήση εργαλείων AI σε επαγγελματικά περιβάλλοντα πρέπει να συνοδεύεται από σαφείς κανόνες σχετικά με το ποια δεδομένα επιτρέπεται να εισάγονται και με ποιον τρόπο χρησιμοποιούνται οι συγκεκριμένες υπηρεσίες.

Παράλληλα, η εξάρτηση από εξωτερικά μοντέλα και υπηρεσίες δημιουργεί ένα πρόσθετο επίπεδο κινδύνου. Η ομάδα ανάπτυξης δεν έχει πάντοτε πλήρη γνώση του τρόπου με τον οποίο δημιουργήθηκε μια απάντηση ούτε μπορεί να θεωρεί δεδομένο ότι διαφορετικές εκδόσεις ενός μοντέλου θα παράγουν ακριβώς τα ίδια αποτελέσματα. Αυτό περιορίζει σε έναν βαθμό την προβλεψιμότητα της διαδικασίας και καθιστά αναγκαία την ύπαρξη μηχανισμών επαλήθευσης πριν από την ενσωμάτωση του αποτελέσματος σε ένα πραγματικό πληροφοριακό σύστημα. Σε εφαρμογές όπου ένα σφάλμα μπορεί να έχει σοβαρές συνέπειες, η ανάγκη αυτή γίνεται ακόμη μεγαλύτερη.

Για τον περιορισμό των παραπάνω κινδύνων, η χρήση της Τεχνητής Νοημοσύνης δεν πρέπει να αντικαθιστά τις καθιερωμένες διαδικασίες ελέγχου του λογισμικού. Ο κώδικας που παράγεται με AI πρέπει να υποβάλλεται στις ίδιες διαδικασίες δοκιμών, αναθεώρησης κώδικα (code review), στατικής ανάλυσης και ελέγχου ασφαλείας που εφαρμόζονται και στον κώδικα που δημιουργείται αποκλειστικά από ανθρώπους. Ιδιαίτερα σε εφαρμογές που διαχειρίζονται προσωπικά δεδομένα, οικονομικές συναλλαγές ή κρίσιμες λειτουργίες, η ανθρώπινη αξιολόγηση παραμένει απαραίτητη. Η Τεχνητή Νοημοσύνη μπορεί να λειτουργήσει ως πρόσθετο επίπεδο υποστήριξης, αλλά δεν αποτελεί από μόνη της μηχανισμό πιστοποίησης της ασφάλειας ή της ορθότητας ενός συστήματος (Pressman & Maxim, 2020).

Η αντιμετώπιση των κινδύνων απαιτεί επίσης την υιοθέτηση της λογικής Security by Design, σύμφωνα με την οποία η ασφάλεια λαμβάνεται υπόψη από τα πρώτα στάδια ανάπτυξης και δεν προστίθεται απλώς μετά την ολοκλήρωση της εφαρμογής. Η αρχή αυτή αποκτά ιδιαίτερη σημασία όταν χρησιμοποιείται AI-generated code, καθώς η ευκολία και η ταχύτητα παραγωγής νέου κώδικα δεν πρέπει να οδηγούν σε χαλάρωση των απαιτήσεων ασφαλείας. Οι προγραμματιστές πρέπει να εξετάζουν την προέλευση και τη λειτουργία των προτεινόμενων λύσεων, να

πραγματοποιούν κατάλληλες δοκιμές και να αποφεύγουν την ενσωμάτωση κώδικα που δεν κατανοούν επαρκώς (McGraw, 2006).

Παρά τους παραπάνω κινδύνους, η σχέση μεταξύ Τεχνητής Νοημοσύνης και ασφάλειας δεν είναι αποκλειστικά αρνητική. Τα ίδια εργαλεία μπορούν να χρησιμοποιηθούν για την αναζήτηση πιθανών αδυναμιών, την επεξήγηση προβληματικών τμημάτων κώδικα και την υποστήριξη της διαδικασίας διόρθωσης. Η Τεχνητή Νοημοσύνη μπορεί επομένως να αποτελέσει μέρος των μηχανισμών ενίσχυσης της ασφάλειας, αρκεί οι προτάσεις της να επαληθεύονται με κατάλληλες τεχνικές και να μην αντιμετωπίζονται ως αλάνθαστες. Η αξία της βρίσκεται κυρίως στη δυνατότητα να λειτουργεί συμπληρωματικά προς τον προγραμματιστή και τα υπάρχοντα εργαλεία ελέγχου.

4.4 Ηθικά και επαγγελματικά ζητήματα

Η ενσωμάτωση της Τεχνητής Νοημοσύνης στη δημιουργία λογισμικού δεν επιφέρει μόνο τεχνικές αλλαγές, αλλά δημιουργεί και σημαντικά ηθικά και επαγγελματικά ζητήματα. Καθώς τα εργαλεία AI αποκτούν τη δυνατότητα να παράγουν κώδικα, να εντοπίζουν σφάλματα, να δημιουργούν τεκμηρίωση και να προτείνουν λύσεις σε προγραμματιστικά προβλήματα, μεταβάλλεται σταδιακά και ο τρόπος με τον οποίο εργάζονται οι προγραμματιστές. Η εξέλιξη αυτή δημιουργεί ερωτήματα σχετικά με την ανθρώπινη ευθύνη, τη διαφάνεια, την προστασία των δεδομένων, την πνευματική ιδιοκτησία και τον μελλοντικό ρόλο των επαγγελματιών του λογισμικού. Επομένως, η αξιολόγηση των συγκεκριμένων τεχνολογιών δεν μπορεί να περιορίζεται μόνο στην ταχύτητα ή στην αποτελεσματικότητά τους, αλλά πρέπει να λαμβάνει υπόψη και τις ευρύτερες συνέπειες που προκαλεί η χρήση τους (Bender et al., 2021).

Ένα δεύτερο σημαντικό ηθικό ζήτημα είναι η ευθύνη για το αποτέλεσμα. Εάν ένας προγραμματιστής χρησιμοποιήσει κώδικα που προτάθηκε από ένα σύστημα Τεχνητής Νοημοσύνης και ο κώδικας αυτός προκαλέσει σοβαρό σφάλμα ή ευπάθεια ασφαλείας, δημιουργείται το ερώτημα ποιος φέρει την ευθύνη. Η χρήση ενός εργαλείου AI δεν απαλλάσσει τον επαγγελματία ή τον οργανισμό από την υποχρέωση να ελέγξει το λογισμικό πριν από τη διάθεσή του. Όπως έχει ήδη αναφερθεί, οι προτάσεις των συστημάτων παραγωγής κώδικα μπορεί να περιλαμβάνουν ευπάθειες, ακόμη και όταν φαίνονται λειτουργικές και τεχνικά πειστικές (Pearce et al., 2022). Για τον λόγο αυτό, η ανθρώπινη εποπτεία αποκτά όχι μόνο τεχνική αλλά και επαγγελματική και ηθική σημασία.

Η έννοια της ευθύνης συνδέεται άμεσα με τη διαφάνεια. Ο προγραμματιστής πρέπει να γνωρίζει πότε ένα τμήμα του έργου έχει παραχθεί με τη βοήθεια AI και να μπορεί, στο μέτρο του δυνατού, να εξηγήσει τον τρόπο λειτουργίας του κώδικα που τελικά ενσωματώνεται στο σύστημα. Η άκριτη χρήση μιας λύσης επειδή προτάθηκε από ένα ισχυρό μοντέλο δεν αποτελεί επαρκή βάση για υπεύθυνη ανάπτυξη λογισμικού. Το ζήτημα αποκτά μεγαλύτερη σημασία σε εφαρμογές στις οποίες τα σφάλματα μπορούν να προκαλέσουν σοβαρές συνέπειες για τους χρήστες ή τους οργανισμούς. Η ανθρώπινη λογοδοσία πρέπει επομένως να παραμένει σαφής ακόμη και όταν μέρος της παραγωγικής διαδικασίας αυτοματοποιείται.

Παράλληλα, η χρήση των συστημάτων αυτών δημιουργεί ζητήματα ιδιωτικότητας και εμπιστευτικότητας. Οι προγραμματιστές μπορεί να εισάγουν σε ένα εργαλείο AI τμήματα πηγαίου κώδικα, περιγραφές εσωτερικών συστημάτων, δεδομένα ή πληροφορίες σχετικά με ένα υπό ανάπτυξη έργο. Εάν το περιεχόμενο αυτό περιλαμβάνει προσωπικά δεδομένα, εμπορικά απόρρητα, κωδικούς πρόσβασης ή άλλο εμπιστευτικό υλικό, η αποστολή του σε μία εξωτερική υπηρεσία μπορεί να δημιουργήσει σοβαρά προβλήματα. Για τον λόγο αυτό, οι οργανισμοί χρειάζεται να θεσπίζουν συγκεκριμένες πολιτικές σχετικά με το ποια δεδομένα μπορούν να χρησιμοποιούνται σε εργαλεία Τεχνητής Νοημοσύνης και υπό ποιες προϋποθέσεις.

Ένα ακόμη ηθικό ζήτημα αφορά τις προκαταλήψεις και τους περιορισμούς των δεδομένων εκπαίδευσης. Τα μεγάλα γλωσσικά μοντέλα εκπαιδεύονται σε πολύ μεγάλες συλλογές δεδομένων και μπορούν να αναπαράγουν προβληματικά χαρακτηριστικά που υπάρχουν σε αυτές. Οι Bender et al. (2021) επισημαίνουν ότι η μεγάλη κλίμακα των γλωσσικών μοντέλων δεν εξαλείφει τους κινδύνους που προέρχονται από τα δεδομένα εκπαίδευσης και τον τρόπο με τον οποίο αυτά συλλέγονται και χρησιμοποιούνται. Στην ανάπτυξη λογισμικού αυτό μπορεί να εκφραστεί, μεταξύ άλλων, μέσω της αναπαραγωγής παρωχημένων, χαμηλής ποιότητας ή προβληματικών πρακτικών που υπάρχουν στον διαθέσιμο κώδικα.

Ιδιαίτερη σημασία έχει επίσης η πνευματική ιδιοκτησία, καθώς τα μοντέλα παραγωγής κώδικα εκπαιδεύονται σε μεγάλες ποσότητες διαθέσιμου ψηφιακού υλικού. Η χρήση κώδικα που μπορεί να προστατεύεται από πνευματικά δικαιώματα κατά την εκπαίδευση μοντέλων έχει δημιουργήσει ερωτήματα σχετικά με τις άδειες χρήσης, τα δικαιώματα των αρχικών δημιουργών και τις προϋποθέσεις υπό τις οποίες μπορούν να αξιοποιούνται τέτοια δεδομένα. Παράλληλα, προκύπτουν ερωτήματα σχετικά με το καθεστώς του ίδιου του αποτελέσματος που παράγεται από ένα σύστημα AI και με τις υποχρεώσεις του χρήστη όταν το αποτέλεσμα εμφανίζει ομοιότητες με ήδη υπάρχοντα κώδικα. Το συγκεκριμένο ζήτημα είναι αρκετά σύνθετο και, για τον λόγο αυτό, εξετάζεται αναλυτικότερα στην επόμενη υποενότητα.

Η πνευματική ιδιοκτησία αποτελεί ένα από τα περισσότερο συζητημένα ζητήματα που συνοδεύουν την ανάπτυξη και χρήση συστημάτων παραγωγικής Τεχνητής Νοημοσύνης. Τα μοντέλα που μπορούν να δημιουργούν προγραμματιστικό κώδικα απαιτούν πολύ μεγάλες ποσότητες δεδομένων για την εκπαίδευσή τους. Στα δεδομένα αυτά μπορεί να περιλαμβάνεται δημόσια διαθέσιμος κώδικας από διαδικτυακά αποθετήρια, ο οποίος όμως δεν είναι απαραίτητα απαλλαγμένος από δικαιώματα ή περιορισμούς χρήσης. Το γεγονός ότι ένα έργο είναι δημόσια προσβάσιμο στο διαδίκτυο δεν σημαίνει αυτομάτως ότι μπορεί να χρησιμοποιηθεί χωρίς να λαμβάνονται υπόψη η άδεια με την οποία διανέμεται και τα δικαιώματα του δημιουργού του.

Το ζήτημα επομένως δεν περιορίζεται στο εάν τα μοντέλα μπορούν τεχνικά να χρησιμοποιήσουν τεράστιες συλλογές κώδικα, αλλά επεκτείνεται στο εάν και υπό ποιες συνθήκες είναι θεμιτή ή νομικά επιτρεπτή η αξιοποίηση προστατευόμενου υλικού για την εκπαίδευσή τους. Η Bender et al. (2021), εξετάζοντας ευρύτερα τα μεγάλα γλωσσικά μοντέλα, υπογραμμίζουν την ανάγκη να εξετάζεται με μεγαλύτερη προσοχή η προέλευση και η τεκμηρίωση των δεδομένων εκπαίδευσης. Η διαφάνεια σχετικά με τα δεδομένα που χρησιμοποιούνται αποκτά επομένως ιδιαίτερη σημασία τόσο για τους δημιουργούς των μοντέλων όσο και για τους επαγγελματίες που χρησιμοποιούν τα αποτελέσματά τους.

Παράλληλα, τίθεται το ερώτημα κατά πόσο οι δημιουργοί λογισμικού πρέπει να έχουν μεγαλύτερο έλεγχο σχετικά με τη χρήση του έργου τους για την εκπαίδευση συστημάτων AI. Στη σχετική συζήτηση περιλαμβάνονται ζητήματα συγκατάθεσης, δυνατότητας εξαίρεσης του έργου από σύνολα δεδομένων εκπαίδευσης και ενδεχόμενης αποζημίωσης των δημιουργών. Πρόκειται για ένα πεδίο στο οποίο οι τεχνολογικές δυνατότητες εξελίχθηκαν ταχύτερα από την πλήρη διαμόρφωση κοινά αποδεκτών πρακτικών, με αποτέλεσμα να παραμένουν ανοιχτά σημαντικά νομικά και ηθικά ερωτήματα.

Εξίσου σημαντικό είναι το ζήτημα του παραγόμενου κώδικα. Ένα εργαλείο Τεχνητής Νοημοσύνης μπορεί να δημιουργήσει μία λύση έπειτα από αίτημα του χρήστη, χωρίς ο τελευταίος να γνωρίζει με ακρίβεια ποια στοιχεία των δεδομένων εκπαίδευσης επηρέασαν το αποτέλεσμα. Σε ορισμένες περιπτώσεις μπορεί να προκύψουν προτάσεις που παρουσιάζουν ομοιότητες με ήδη υπάρχοντα αποσπάσματα κώδικα. Αυτό δημιουργεί πρακτικά ερωτήματα για έναν επαγγελματία προγραμματιστή, καθώς η ενσωμάτωση κώδικα σε ένα εμπορικό προϊόν πρέπει να γίνεται με τρόπο συμβατό με τις σχετικές άδειες και τις υποχρεώσεις του οργανισμού.

Για τον λόγο αυτό, η χρήση AI-generated code σε επαγγελματικά έργα δεν πρέπει να αντιμετωπίζεται ως μία διαδικασία χωρίς νομικούς περιορισμούς. Οι οργανισμοί χρειάζεται να διαθέτουν πολιτικές για τον τρόπο χρήσης των συγκεκριμένων εργαλείων, ενώ οι προγραμματιστές πρέπει να γνωρίζουν ότι η παραγωγή μιας απάντησης από ένα σύστημα AI δεν αποτελεί από μόνη της εγγύηση ότι αυτή μπορεί να χρησιμοποιηθεί χωρίς περαιτέρω έλεγχο. Η τεχνική αξιολόγηση του κώδικα χρειάζεται επομένως να συνοδεύεται, όπου απαιτείται, από έλεγχο των αδειών και των πιθανών ζητημάτων πνευματικής ιδιοκτησίας.

4.5 Σύγχρονοι προβληματισμοί για την Τεχνητή Νοημοσύνη

Η ταχεία εξέλιξη της Τεχνητής Νοημοσύνης και ιδιαίτερα των μεγάλων γλωσσικών μοντέλων έχει δημιουργήσει έναν ευρύτερο διάλογο σχετικά με τις συνέπειες της αυξανόμενης χρήσης τους. Παρότι οι προηγούμενες ενότητες ανέδειξαν σημαντικά οφέλη για την ανάπτυξη λογισμικού, όπως η αύξηση της παραγωγικότητας, η υποστήριξη της συγγραφής κώδικα και η αυτοματοποίηση επιμέρους διαδικασιών, η συνεχής επέκταση των συστημάτων AI δημιουργεί νέους τεχνολογικούς, κοινωνικούς και περιβαλλοντικούς προβληματισμούς. Η αξιολόγηση της Τεχνητής Νοημοσύνης δεν μπορεί επομένως να βασίζεται αποκλειστικά στις δυνατότητες ή στην απόδοση

των μοντέλων, αλλά χρειάζεται να λαμβάνει υπόψη το κόστος ανάπτυξης και λειτουργίας τους, τις επιπτώσεις στην εργασία και τη βιωσιμότητα της συνεχούς αύξησης της υπολογιστικής ισχύος που απαιτούν (Bender et al., 2021).

Ένας από τους σημαντικότερους σύγχρονους προβληματισμούς αφορά την αυξανόμενη κλίμακα των μοντέλων Τεχνητής Νοημοσύνης. Η πρόοδος των τελευταίων ετών έχει συνδεθεί σε σημαντικό βαθμό με την ανάπτυξη μεγαλύτερων μοντέλων, την αξιοποίηση πολύ μεγάλων συνόλων δεδομένων και τη χρήση ισχυρών υπολογιστικών υποδομών. Ωστόσο, η αύξηση της υπολογιστικής ισχύος δεν είναι χωρίς κόστος. Η εκπαίδευση σύνθετων μοντέλων βαθιάς μάθησης μπορεί να απαιτεί σημαντική κατανάλωση ηλεκτρικής ενέργειας και οικονομικών πόρων, γεγονός που δημιουργεί ερωτήματα σχετικά με το κατά πόσο η συνεχής αύξηση του μεγέθους αποτελεί μία βιώσιμη κατεύθυνση για την εξέλιξη της AI (Strubell et al., 2019).

Στο πλαίσιο αυτό έχει αναπτυχθεί η προσέγγιση της Πράσινης Τεχνητής Νοημοσύνης (Green AI). Οι Schwartz et al. (2020) διακρίνουν την προσέγγιση αυτή από μία λογική σύμφωνα με την οποία το ενδιαφέρον επικεντρώνεται κυρίως στη βελτίωση της ακρίβειας ή της απόδοσης των μοντέλων, ακόμη και όταν απαιτείται σημαντικά μεγαλύτερη υπολογιστική ισχύς. Η Green AI δίνει μεγαλύτερη έμφαση στην αποδοτικότητα και υποστηρίζει ότι η πρόοδος ενός συστήματος πρέπει να εξετάζεται και σε σχέση με τους πόρους που απαιτούνται για την επίτευξή της. Με αυτόν τον τρόπο, ένα αποτελεσματικότερο μοντέλο δεν είναι απαραίτητα μόνο εκείνο που επιτυγχάνει την υψηλότερη απόδοση, αλλά και εκείνο που μπορεί να επιτύχει ικανοποιητικά αποτελέσματα χρησιμοποιώντας λιγότερους υπολογιστικούς και ενεργειακούς πόρους (Schwartz et al., 2020).

Η συγκεκριμένη προσέγγιση έχει ιδιαίτερη σημασία και για τη χρήση της Τεχνητής Νοημοσύνης στη δημιουργία λογισμικού. Ένας προγραμματιστής που χρησιμοποιεί ένα AI coding assistant βλέπει κυρίως το τελικό αποτέλεσμα, όπως μία πρόταση κώδικα ή μία απάντηση σε ένα τεχνικό ερώτημα. Πίσω από αυτή τη φαινομενικά απλή αλληλεπίδραση, όμως, βρίσκεται μία ευρύτερη υπολογιστική υποδομή που είναι απαραίτητη για τη λειτουργία του μοντέλου. Καθώς τέτοια εργαλεία ενσωματώνονται σταδιακά στην καθημερινή εργασία μεγάλου αριθμού προγραμματιστών, η περιβαλλοντική συζήτηση δεν αφορά πλέον μόνο την αρχική εκπαίδευση των μοντέλων, αλλά και τη λειτουργία τους κατά τη συνεχή χρήση τους.

Επομένως, είναι σημαντικό να γίνεται διάκριση ανάμεσα στην εκπαίδευση (training) και στη χρήση ή εξαγωγή συμπερασμάτων (inference) ενός μοντέλου. Η εκπαίδευση μπορεί να αποτελεί μία ιδιαίτερα απαιτητική υπολογιστική διαδικασία, όμως μετά την ολοκλήρωσή της το μοντέλο μπορεί να χρησιμοποιηθεί πολλές φορές. Κάθε αίτημα προς ένα μεγάλο γλωσσικό μοντέλο απαιτεί επίσης υπολογιστική επεξεργασία. Συνεπώς, όταν ένα εργαλείο χρησιμοποιείται σε πολύ μεγάλη κλίμακα, το συνολικό περιβαλλοντικό του αποτύπωμα δεν εξαρτάται αποκλειστικά από την αρχική εκπαίδευση αλλά και από τη συχνότητα και τον τρόπο με τον οποίο χρησιμοποιείται.

Πέρα από την κατανάλωση ηλεκτρικής ενέργειας και τις σχετικές εκπομπές άνθρακα, τα τελευταία χρόνια έχει αναδειχθεί και το ζήτημα της κατανάλωσης νερού από τις υποδομές που υποστηρίζουν τα συστήματα Τεχνητής Νοημοσύνης. Τα κέντρα δεδομένων χρειάζονται συστήματα ψύξης, ενώ η παραγωγή ηλεκτρικής ενέργειας και η λειτουργία των σχετικών υποδομών μπορούν επίσης να συνδέονται με χρήση υδάτινων πόρων. Οι Li et al. (2023) ανέδειξαν αυτή τη λιγότερο εμφανή διάσταση του περιβαλλοντικού αποτυπώματος της AI και υποστήριξαν ότι η αξιολόγηση της βιωσιμότητας πρέπει να εξετάζει το υδατικό αποτύπωμα μαζί με το ενεργειακό και το ανθρακικό αποτύπωμα.

Χαρακτηριστικά, οι Li et al. (2023) εκτίμησαν ότι η εκπαίδευση του GPT-3 σε κέντρα δεδομένων της Microsoft στις Ηνωμένες Πολιτείες μπορούσε να συνδέεται με περίπου 700.000 λίτρα άμεσης κατανάλωσης γλυκού νερού για την ψύξη. Οι ίδιοι συγγραφείς τονίζουν ότι το πραγματικό υδατικό αποτύπωμα επηρεάζεται σημαντικά από τον τόπο και τον χρόνο λειτουργίας της υπολογιστικής υποδομής. Επομένως, τέτοιες αριθμητικές εκτιμήσεις δεν πρέπει να γενικεύονται σε όλα τα μοντέλα ή σε κάθε κέντρο δεδομένων, αλλά αναδεικνύουν μία περιβαλλοντική διάσταση που χρειάζεται να συνυπολογίζεται στη συζήτηση για τη βιώσιμη ανάπτυξη της AI (Li et al., 2023).

Η περιβαλλοντική επίδραση συνδέεται επίσης με την υλική υποδομή που απαιτείται για την ανάπτυξη της Τεχνητής Νοημοσύνης. Τα μεγάλα μοντέλα δεν λειτουργούν ανεξάρτητα από φυσικούς πόρους, αλλά απαιτούν εξειδικευμένους επεξεργαστές, διακομιστές, κέντρα δεδομένων, δίκτυα και συστήματα ψύξης. Συνεπώς, η συζήτηση για το περιβαλλοντικό αποτύπωμα δεν πρέπει να περιορίζεται αποκλειστικά στην ηλεκτρική ενέργεια που

καταναλώνεται κατά την εκτέλεση ενός μοντέλου. Χρειάζεται να εξετάζεται ευρύτερα ο κύκλος ζωής των τεχνολογικών υποδομών που καθιστούν δυνατή την ανάπτυξη και τη λειτουργία του.

Η βιωσιμότητα συνδέεται επίσης με το ζήτημα της διαφάνειας. Για να είναι δυνατή η ουσιαστική σύγκριση διαφορετικών συστημάτων AI, απαιτούνται πληροφορίες όχι μόνο για την απόδοσή τους αλλά και για τους πόρους που καταναλώνουν. Ωστόσο, δεν δημοσιοποιούνται πάντοτε αναλυτικά στοιχεία για την ενέργεια, το νερό και τις συνολικές υποδομές που απαιτούνται για την εκπαίδευση και τη λειτουργία εμπορικών μοντέλων. Η περιορισμένη πρόσβαση σε τέτοια δεδομένα δυσχεραίνει την ακριβή αποτίμηση του πραγματικού περιβαλλοντικού κόστους και καθιστά σημαντική την ανάπτυξη κοινών μεθόδων μέτρησης και αναφοράς (Li et al., 2023).

Ένας διαφορετικός σύγχρονος προβληματισμός αφορά τη συγκέντρωση της τεχνολογικής ισχύος. Η εκπαίδευση ιδιαίτερα μεγάλων μοντέλων απαιτεί σημαντικούς οικονομικούς και υπολογιστικούς πόρους, οι οποίοι δεν είναι εξίσου διαθέσιμοι σε πανεπιστήμια, μικρές επιχειρήσεις, ανεξάρτητους ερευνητές και μεγάλες τεχνολογικές εταιρείες. Η υψηλή απαίτηση σε πόρους μπορεί επομένως να δημιουργήσει εμπόδια συμμετοχής στην έρευνα και ανάπτυξη προηγμένων συστημάτων AI. Οι Schwartz et al. (2020) συνδέουν την υπολογιστική αποδοτικότητα και με τη μεγαλύτερη προσβασιμότητα της έρευνας, καθώς η ανάγκη για ολοένα μεγαλύτερους πόρους μπορεί να περιορίζει τη δυνατότητα συμμετοχής όσων δεν διαθέτουν αντίστοιχες υποδομές.

Ευρύτερα, η Bender et al. (2021) υπογραμμίζει την ανάγκη να μην αξιολογούνται τα μεγάλα γλωσσικά μοντέλα μόνο βάσει των εντυπωσιακών αποτελεσμάτων που μπορούν να παράγουν. Ιδιαίτερη σημασία έχουν τα δεδομένα στα οποία εκπαιδεύονται, οι προκαταλήψεις που μπορεί να αναπαράγουν, η αδιαφάνεια που συνοδεύει ορισμένες διαδικασίες ανάπτυξης και το περιβαλλοντικό κόστος της συνεχούς αύξησης του μεγέθους τους. Η προσέγγιση αυτή είναι ιδιαίτερα χρήσιμη για την παρούσα εργασία, επειδή δείχνει ότι η τεχνολογική εξέλιξη της AI στη δημιουργία λογισμικού πρέπει να αξιολογείται παράλληλα με τις ευρύτερες συνέπειές της.

Κεφάλαιο 5: Το μέλλον της τεχνητής νοημοσύνης στη μηχανική λογισμικού

5.1 Εξέλιξη των εργαλείων Τεχνητής Νοημοσύνης για την ανάπτυξη λογισμικού

Η αξιοποίηση της Τεχνητής Νοημοσύνης στην ανάπτυξη λογισμικού εξελίσσεται με ιδιαίτερα γρήγορο ρυθμό και επηρεάζει σταδιακά τον τρόπο με τον οποίο οργανώνεται και πραγματοποιείται η εργασία των προγραμματιστών. Τα πρώτα εργαλεία αυτοματοποίησης επικεντρώνονταν κυρίως σε συγκεκριμένες και περιορισμένες λειτουργίες, όπως η αυτόματη συμπλήρωση εντολών, η στατική ανάλυση κώδικα και ο εντοπισμός συντακτικών σφαλμάτων. Η εμφάνιση των μεγάλων γλωσσικών μοντέλων (Large Language Models - LLMs) επέκτεινε σημαντικά αυτές τις δυνατότητες, καθώς τα σύγχρονα συστήματα μπορούν πλέον να παράγουν κώδικα, να εξηγούν υπάρχοντα προγράμματα, να προτείνουν διορθώσεις και να υποστηρίζουν διαφορετικές εργασίες της μηχανικής λογισμικού (Jin et al., 2024). Η εξέλιξη αυτή δείχνει μία σταδιακή μετάβαση από απλά εργαλεία αυτοματοποίησης σε περισσότερο ολοκληρωμένα συστήματα υποστήριξης του προγραμματιστή.

Σημαντικό στάδιο αυτής της εξέλιξης αποτέλεσε η ανάπτυξη μοντέλων που μπορούν να επεξεργάζονται τόσο φυσική γλώσσα όσο και προγραμματιστικό κώδικα. Η δυνατότητα αυτή επιτρέπει στον χρήστη να περιγράφει μία εργασία ή ένα προγραμματιστικό πρόβλημα με φυσική γλώσσα και το σύστημα να παράγει μία πιθανή λύση σε μορφή κώδικα. Με τον τρόπο αυτό αλλάζει σταδιακά και η μορφή της αλληλεπίδρασης μεταξύ ανθρώπου και υπολογιστή κατά τη δημιουργία λογισμικού. Ο προγραμματιστής δεν χρειάζεται πάντοτε να ξεκινά από ένα κενό αρχείο και να γράφει κάθε εντολή μόνος του, αλλά μπορεί να χρησιμοποιεί την Τεχνητή Νοημοσύνη για τη δημιουργία ενός προσχεδίου, το οποίο στη συνέχεια ελέγχει, προσαρμόζει και ενσωματώνει στο έργο του.

Χαρακτηριστικό παράδειγμα αποτελεί το SWE-bench, το οποίο σχεδιάστηκε για να αξιολογεί κατά πόσο τα γλωσσικά μοντέλα μπορούν να επιλύσουν πραγματικά προβλήματα που προέρχονται από έργα λογισμικού. Σε αντίθεση με μία απλή άσκηση προγραμματισμού, σε

τέτοιες περιπτώσεις το σύστημα πρέπει να εξετάσει τον υπάρχοντα κώδικα, να κατανοήσει το πρόβλημα και να πραγματοποιήσει τις απαραίτητες αλλαγές. Η αρχική μελέτη του SWE-bench χρησιμοποίησε 2.294 πραγματικά ζητήματα από 12 δημοφιλή αποθετήρια Python και έδειξε ότι αυτού του είδους οι εργασίες αποτελούσαν σημαντική πρόκληση για τα μοντέλα που εξετάστηκαν (Jimenez et al., 2024). Το παράδειγμα αυτό είναι σημαντικό επειδή δείχνει ότι η αξιολόγηση των εργαλείων AI μετακινείται από την απλή παραγωγή κώδικα προς περισσότερο ρεαλιστικές συνθήκες ανάπτυξης λογισμικού.

Η κατεύθυνση αυτή οδηγεί σταδιακά στην ανάπτυξη συστημάτων που μπορούν να αξιοποιούν μεγαλύτερο μέρος του περιβάλλοντος ενός έργου. Αντί το μοντέλο να λαμβάνει μόνο μία ερώτηση και να επιστρέφει ένα απόσπασμα κώδικα, τα νεότερα συστήματα επιχειρούν να εξετάζουν περισσότερες πληροφορίες, να εντοπίζουν τα αρχεία που σχετίζονται με ένα πρόβλημα και να προτείνουν αλλαγές που λαμβάνουν υπόψη τη δομή του υπάρχοντος λογισμικού. Πρόκειται για μία ιδιαίτερα σημαντική εξέλιξη, καθώς η πραγματική αξία της AI στη μηχανική λογισμικού εξαρτάται σε μεγάλο βαθμό από το κατά πόσο μπορεί να λειτουργήσει μέσα σε ένα πραγματικό και σύνθετο έργο και όχι μόνο σε απομονωμένες προγραμματιστικές ασκήσεις (Liu et al., 2024).

Μία ακόμη σημαντική εξέλιξη αφορά την ενσωμάτωση της AI στα εργαλεία που χρησιμοποιεί καθημερινά ο προγραμματιστής. Η αποτελεσματικότητα ενός συστήματος δεν εξαρτάται μόνο από την ικανότητά του να παράγει μία σωστή απάντηση, αλλά και από το πόσο ομαλά μπορεί να ενταχθεί στην υπάρχουσα διαδικασία ανάπτυξης. Η σύνδεση με περιβάλλοντα ανάπτυξης, αποθετήρια κώδικα, εργαλεία δοκιμών και άλλες τεχνικές υποδομές μπορεί να επιτρέψει στην AI να αποκτήσει μεγαλύτερη εικόνα για το έργο στο οποίο χρησιμοποιείται. Με τον τρόπο αυτό, η αλληλεπίδραση μπορεί να γίνει περισσότερο ουσιαστική σε σχέση με ένα απλό σύστημα στο οποίο ο προγραμματιστής αντιγράφει χειροκίνητα ένα απόσπασμα κώδικα και ζητά μία απάντηση (Liu et al., 2024).

Η εξέλιξη αυτή συνδέεται άμεσα και με την αυξανόμενη δυνατότητα των συστημάτων να πραγματοποιούν περισσότερα από ένα διαδοχικά βήματα. Ένα απλό εργαλείο παραγωγής κώδικα μπορεί να δεχτεί μία οδηγία και να δημιουργήσει μία απάντηση. Ένα περισσότερο προηγμένο σύστημα μπορεί να χρειάζεται αρχικά να αναλύσει το πρόβλημα, στη συνέχεια να εντοπίσει το κατάλληλο τμήμα του έργου, να προτείνει ή να πραγματοποιήσει μία αλλαγή, να εκτελέσει δοκιμές και να χρησιμοποιήσει τα αποτελέσματά τους για να αποφασίσει εάν απαιτείται νέα διόρθωση. Η δυνατότητα αυτή μεταφέρει σταδιακά την Τεχνητή Νοημοσύνη από την απλή παραγωγή περιεχομένου προς την εκτέλεση περισσότερο σύνθετων διαδικασιών ανάπτυξης (Liu et al., 2024).

Στο πλαίσιο αυτό εμφανίζεται η τάση ανάπτυξης των AI agents, δηλαδή συστημάτων που συνδυάζουν τις δυνατότητες ενός μεγάλου γλωσσικού μοντέλου με μηχανισμούς σχεδιασμού ενεργειών και χρήσης πρόσθετων εργαλείων. Ένας τέτοιος agent δεν περιορίζεται απαραίτητα στην παραγωγή μίας απάντησης, αλλά μπορεί να επιχειρεί μία σειρά ενεργειών για την επίτευξη ενός συγκεκριμένου στόχου. Η συγκεκριμένη κατεύθυνση θεωρείται σημαντική για τη μελλοντική εξέλιξη της μηχανικής λογισμικού, επειδή δημιουργεί την προοπτική ανάθεσης περισσότερο σύνθετων εργασιών σε συστήματα Τεχνητής Νοημοσύνης (Liu et al., 2024).

Παρά τη γρήγορη πρόοδο, η έρευνα δείχνει επίσης ότι υπάρχει σημαντική διαφορά μεταξύ της επιτυχίας ενός μοντέλου σε περιορισμένες προγραμματιστικές εργασίες και της αξιόπιστης λειτουργίας του σε μεγάλα πραγματικά έργα. Η αντιμετώπιση προβλημάτων που απαιτούν κατανόηση πολλών αρχείων, συνδυασμό διαφορετικών πληροφοριών και πραγματοποίηση διαδοχικών αλλαγών παραμένει περισσότερο απαιτητική (Jimenez et al., 2024). Επομένως, οι μελλοντικές εξελίξεις θα πρέπει να αξιολογούνται με βάση πραγματικές εργασίες μηχανικής λογισμικού και όχι αποκλειστικά από την ικανότητα ενός μοντέλου να δημιουργεί εντυπωσιακά αποσπάσματα κώδικα.

Η μελλοντική εξέλιξη των εργαλείων αναμένεται επομένως να εξαρτηθεί όχι μόνο από τη δημιουργία μεγαλύτερων ή ισχυρότερων μοντέλων, αλλά και από τη βελτίωση της αξιοπιστίας, της κατανόησης του πλαισίου και της δυνατότητας ελέγχου των ενεργειών τους. Ένα αποτελεσματικό εργαλείο για πραγματική ανάπτυξη λογισμικού χρειάζεται να μπορεί να εργάζεται με μεγαλύτερα έργα, να αναγνωρίζει σχέσεις και εξαρτήσεις μεταξύ διαφορετικών στοιχείων, να αξιοποιεί τα αποτελέσματα των δοκιμών και να προσαρμόζει τις προτάσεις του όταν μία αρχική λύση αποτυγχάνει. Παράλληλα, απαιτούνται μηχανισμοί που θα επιτρέπουν στον

προγραμματιστή να κατανοεί και να ελέγχει τις αλλαγές που προτείνονται πριν αυτές ενσωματωθούν στο τελικό σύστημα.

5.2 AI Agents και περισσότερο αυτόνομη ανάπτυξη λογισμικού

Μία από τις σημαντικότερες κατευθύνσεις εξέλιξης της Τεχνητής Νοημοσύνης στη μηχανική λογισμικού είναι η ανάπτυξη των AI agents, δηλαδή συστημάτων που δεν περιορίζονται στην παραγωγή μιας μεμονωμένης απάντησης ή ενός αποσπάσματος κώδικα, αλλά μπορούν να οργανώνουν και να εκτελούν μία σειρά ενεργειών για την ολοκλήρωση μιας περισσότερο σύνθετης εργασίας. Η εξέλιξη αυτή διαφοροποιείται από τη λειτουργία των απλών εργαλείων παραγωγής κώδικα, καθώς οι agents μπορούν να συνδυάζουν τις δυνατότητες των μεγάλων γλωσσικών μοντέλων με την πρόσβαση σε εξωτερικούς πόρους και εργαλεία (Liu et al., 2024).

Η βασική διαφορά ανάμεσα σε έναν AI agent και ένα συμβατικό εργαλείο παραγωγής κώδικα βρίσκεται επομένως στον βαθμό αυτονομίας που μπορεί να αποκτήσει το σύστημα. Ένα συμβατικό εργαλείο δέχεται συνήθως μία οδηγία και επιστρέφει μία πρόταση κώδικα, αφήνοντας στον προγραμματιστή την ευθύνη για τα επόμενα βήματα. Αντίθετα, ένας agent μπορεί να λάβει έναν ευρύτερο στόχο, να τον αναλύσει σε μικρότερες εργασίες και να πραγματοποιήσει διαδοχικές ενέργειες για την επίτευξή του, χρησιμοποιώντας παράλληλα πληροφορίες που προκύπτουν κατά τη διάρκεια της διαδικασίας (Liu et al., 2024).

Στη μηχανική λογισμικού, η δυνατότητα αυτή μπορεί να έχει σημαντικές εφαρμογές. Ένας agent μπορεί, για παράδειγμα, να χρειαστεί να εντοπίσει τα αρχεία που σχετίζονται με ένα συγκεκριμένο πρόβλημα, να εξετάσει τον υπάρχοντα κώδικα, να προτείνει ή να πραγματοποιήσει μία αλλαγή και στη συνέχεια να εκτελέσει τις διαθέσιμες δοκιμές. Εάν οι δοκιμές αποτύχουν, το αποτέλεσμα μπορεί να αποτελέσει νέα πληροφορία για το σύστημα, ώστε να επανεξετάσει την αρχική λύση και να πραγματοποιήσει νέα προσπάθεια (Yang et al., 2024).

Αυτή η επαναληπτική λειτουργία είναι ιδιαίτερα σημαντική, επειδή η ανάπτυξη λογισμικού δεν αποτελεί μία διαδικασία που ολοκληρώνεται συνήθως με μία μόνο ενέργεια. Η αντιμετώπιση ενός προβλήματος μπορεί να απαιτεί διαδοχική ανάλυση, τροποποίηση, εκτέλεση, έλεγχο και διόρθωση του κώδικα. Η δυνατότητα των agents να αλληλεπιδρούν με το περιβάλλον και να προσαρμόζουν τις επόμενες ενέργειές τους σύμφωνα με τα αποτελέσματα προηγούμενων ενεργειών αποτελεί ένα βασικό χαρακτηριστικό που τους διαφοροποιεί από την απλή χρήση ενός γλωσσικού μοντέλου (Yang et al., 2024).

Ιδιαίτερα σημαντική είναι επίσης η δυνατότητα των AI agents να χρησιμοποιούν εξωτερικά εργαλεία. Ένα μεγάλο γλωσσικό μοντέλο από μόνο του παράγει κυρίως απαντήσεις με βάση τις πληροφορίες που του παρέχονται. Ένας agent μπορεί αντίθετα να συνδυάζει το μοντέλο με πρόσθετες λειτουργίες και να αλληλεπιδρά με ένα περιβάλλον ανάπτυξης, αρχεία κώδικα και μηχανισμούς εκτέλεσης ή δοκιμών. Με τον τρόπο αυτό, η Τεχνητή Νοημοσύνη μετακινείται από την απλή παροχή συμβουλών προς την πραγματοποίηση συγκεκριμένων ενεργειών που αποτελούν μέρος της διαδικασίας ανάπτυξης λογισμικού (Liu et al., 2024).

Χαρακτηριστικό παράδειγμα αυτής της προσέγγισης αποτελεί το SWE-agent, ένα σύστημα που αναπτύχθηκε με στόχο να επιτρέψει σε γλωσσικά μοντέλα να αντιμετωπίζουν πραγματικές εργασίες μηχανικής λογισμικού. Το σύστημα διαθέτει ειδικά σχεδιασμένη διεπαφή μέσω της οποίας ο agent μπορεί να περιηγείται σε αποθετήρια, να δημιουργεί και να επεξεργάζεται αρχεία κώδικα και να εκτελεί δοκιμές και άλλα προγράμματα. Η μελέτη του SWE-agent έδειξε ότι ο τρόπος με τον οποίο ένας agent αλληλεπιδρά με το υπολογιστικό περιβάλλον επηρεάζει ουσιαστικά την ικανότητά του να αντιμετωπίζει σύνθετες εργασίες λογισμικού (Yang et al., 2024).

Η ανάπτυξη τέτοιων συστημάτων συνδέεται άμεσα με την ανάγκη για καλύτερη κατανόηση του ευρύτερου πλαισίου ενός έργου λογισμικού. Ένα πραγματικό σφάλμα μπορεί να μην βρίσκεται αποκλειστικά σε μία γραμμή ή σε μία συνάρτηση, αλλά να σχετίζεται με διαφορετικές κλάσεις, συναρτήσεις ή αρχεία. Ο agent πρέπει επομένως να εντοπίζει ποιες πληροφορίες είναι σχετικές με το πρόβλημα και να κατανοεί σε έναν βαθμό τις σχέσεις μεταξύ διαφορετικών τμημάτων του έργου πριν προχωρήσει σε αλλαγές. Πρόκειται για μία σημαντικά πιο απαιτητική εργασία σε σχέση με την παραγωγή μιας ανεξάρτητης συνάρτησης από μία σύντομη περιγραφή (Jimenez et al., 2024).

Η πολυπλοκότητα αυτή έχει αναδειχθεί μέσα από το SWE-bench, ένα πλαίσιο αξιολόγησης που βασίζεται σε πραγματικά προβλήματα λογισμικού από αποθετήρια ανοικτού κώδικα. Το

αρχικό SWE-bench περιλαμβάνει 2.294 προβλήματα από 12 δημοφιλή αποθετήρια Python και απαιτεί από τα συστήματα να επεξεργαστούν υπάρχουσες βάσεις κώδικα και να δημιουργήσουν κατάλληλες διορθώσεις. Πολλά από τα προβλήματα απαιτούν αλλαγές που συνδέονται με περισσότερες από μία συναρτήσεις, κλάσεις ή αρχεία, γεγονός που τα καθιστά περισσότερο αντιπροσωπευτικά πραγματικών εργασιών μηχανικής λογισμικού σε σχέση με απλές ασκήσεις παραγωγής κώδικα (Jimenez et al., 2024).

Τα αποτελέσματα της αρχικής μελέτης του SWE-bench ανέδειξαν επίσης τη δυσκολία που παρουσιάζουν οι πραγματικές εργασίες λογισμικού για τα γλωσσικά μοντέλα. Τα μοντέλα που εξετάστηκαν στην αρχική έρευνα μπορούσαν να επιλύσουν μόνο περιορισμένο ποσοστό των προβλημάτων, γεγονός που έδειξε ότι η επιτυχία σε μικρές προγραμματιστικές ασκήσεις δεν συνεπάγεται αντίστοιχη ικανότητα αντιμετώπισης ενός πραγματικού έργου λογισμικού. Για την επίλυση τέτοιων προβλημάτων απαιτούνται μεγαλύτερο πλαίσιο πληροφοριών, σύνθετη συλλογιστική και δυνατότητα αλληλεπίδρασης με το περιβάλλον εκτέλεσης (Jimenez et al., 2024).

Η ανάπτυξη των AI agents δημιουργεί επομένως την προοπτική μιας περισσότερο αυτόνομης διαδικασίας ανάπτυξης λογισμικού. Στο πλαίσιο αυτό, ο προγραμματιστής μπορεί να καθορίζει έναν στόχο σε υψηλότερο επίπεδο και το σύστημα να αναλαμβάνει μέρος των επιμέρους ενεργειών που απαιτούνται για την επίτευξή του. Αντί να ζητείται ξεχωριστά η παραγωγή κάθε τμήματος κώδικα, ένας agent μπορεί να αναλύσει το πρόβλημα, να εντοπίσει τα σχετικά σημεία του έργου, να επιχειρήσει μία λύση και να ελέγξει το αποτέλεσμα πριν το παρουσιάσει στον χρήστη (Liu et al., 2024).

Μία ακόμη κατεύθυνση που εξετάζεται είναι η χρήση πολλαπλών agents, οι οποίοι μπορούν να συνεργάζονται για την ολοκλήρωση μίας σύνθετης εργασίας. Η λογική αυτή βασίζεται στην κατανομή της εργασίας μεταξύ διαφορετικών ρόλων, έτσι ώστε κάθε agent να αναλαμβάνει ένα συγκεκριμένο μέρος της διαδικασίας. Η συνεργασία μεταξύ διαφορετικών agents μπορεί θεωρητικά να επιτρέψει την καλύτερη οργάνωση σύνθετων εργασιών, ιδιαίτερα όταν αυτές περιλαμβάνουν διαφορετικά στάδια και απαιτούν διαφορετικές μορφές επεξεργασίας πληροφοριών (Liu et al., 2024).

Οι εξελίξεις αυτές δημιουργούν την πιθανότητα να αυτοματοποιηθούν σταδιακά περισσότερες εργασίες που σήμερα πραγματοποιούνται από προγραμματιστές. Επαναλαμβανόμενες ή σαφώς προσδιορισμένες δραστηριότητες, όπως ορισμένες διορθώσεις κώδικα, η παραγωγή βασικών δοκιμών ή η πραγματοποίηση περιορισμένων αλλαγών σε υπάρχοντα έργα, μπορούν να αποτελέσουν κατάλληλες περιπτώσεις για μεγαλύτερη αυτοματοποίηση. Η έρευνα πάνω σε συστήματα όπως το SWE-agent δείχνει ήδη τη δυνατότητα ενός agent να επεξεργάζεται αρχεία, να εκτελεί προγράμματα και να επιχειρεί την επίλυση πραγματικών προβλημάτων λογισμικού (Yang et al., 2024).

Ωστόσο, η έννοια της αυτόνομης ανάπτυξης λογισμικού πρέπει να αντιμετωπίζεται με προσοχή. Η δυνατότητα ενός agent να πραγματοποιεί πολλές ενέργειες δεν εξασφαλίζει ότι οι αποφάσεις του είναι πάντοτε σωστές. Ένα σφάλμα στην αρχική ερμηνεία του προβλήματος μπορεί να επηρεάσει τα επόμενα βήματα, ενώ μία αλλαγή που διορθώνει ένα συγκεκριμένο πρόβλημα μπορεί να δημιουργήσει δυσλειτουργία σε διαφορετικό μέρος του συστήματος. Η δυσκολία των πραγματικών προβλημάτων που περιλαμβάνονται στο SWE-bench επιβεβαιώνει ότι η αξιόπιστη επίλυση σύνθετων εργασιών λογισμικού εξακολουθεί να αποτελεί σημαντική πρόκληση (Jimenez et al., 2024).

Επιπλέον, οι πραγματικές απαιτήσεις ενός έργου δεν αποτυπώνονται πάντοτε πλήρως στον πηγαίο κώδικα ή στην περιγραφή ενός προβλήματος. Μπορεί να υπάρχουν επιχειρησιακοί περιορισμοί, ανάγκες χρηστών, κανόνες ενός οργανισμού ή τεχνικές αποφάσεις που δεν είναι εμφανείς σε ένα μοντέλο. Η αντιμετώπιση πραγματικών εργασιών μηχανικής λογισμικού απαιτεί συχνά συνδυασμό τεχνικής γνώσης και κατανόησης του ευρύτερου πλαισίου στο οποίο λειτουργεί μία εφαρμογή, γεγονός που περιορίζει την πλήρη αυτονομία των σημερινών συστημάτων (Jimenez et al., 2024).

Όσο αυξάνεται η αυτονομία των agents, τόσο μεγαλύτερη σημασία αποκτά και η ασφάλεια των ενεργειών τους. Ένα εργαλείο που απλώς προτείνει ένα απόσπασμα κώδικα έχει διαφορετικό επίπεδο πρόσβασης από έναν agent που μπορεί να επεξεργάζεται αρχεία, να εκτελεί εντολές ή να χρησιμοποιεί άλλα εργαλεία. Για τον λόγο αυτό, η σχεδίαση των LLM-based agents πρέπει να λαμβάνει υπόψη τον τρόπο με τον οποίο τα συστήματα αλληλεπιδρούν με εξωτερικούς πόρους και τα όρια που τίθενται στις ενέργειές τους (Liu et al., 2024).

Αντίστοιχα, ιδιαίτερη σημασία αποκτούν οι μηχανισμοί ελέγχου και ανθρώπινης παρέμβασης. Ένας agent μπορεί να αναλαμβάνει την εκτέλεση συγκεκριμένων βημάτων, αλλά ο άνθρωπος πρέπει να έχει τη δυνατότητα να εξετάζει τις αλλαγές και να εγκρίνει κρίσιμες ενέργειες πριν αυτές επηρεάσουν ένα πραγματικό σύστημα. Η ανάπτυξη του SWE-agent δείχνει ότι η σχεδίαση της διεπαφής μεταξύ του agent και του υπολογιστικού περιβάλλοντος αποτελεί βασικό παράγοντα για τη συμπεριφορά και την απόδοσή του, στοιχείο που αναδεικνύει τη σημασία του τρόπου με τον οποίο ελέγχεται η πρόσβαση του συστήματος στα διαθέσιμα εργαλεία (Yang et al., 2024).

Παράλληλα, απαιτούνται κατάλληλοι τρόποι αξιολόγησης της απόδοσης των agents. Δεν είναι αρκετό να εξετάζεται μόνο εάν ο τελικός κώδικας μπορεί να εκτελεστεί. Πρέπει να αξιολογείται εάν το σύστημα πραγματοποίησε τις κατάλληλες αλλαγές, εάν διατήρησε τις υπόλοιπες λειτουργίες του έργου και εάν η λύση ανταποκρίνεται πραγματικά στο πρόβλημα που έπρεπε να αντιμετωπιστεί. Η χρήση πραγματικών ζητημάτων λογισμικού στο SWE-bench αποτελεί ένα σημαντικό βήμα προς περισσότερο ρεαλιστικές μορφές αξιολόγησης των δυνατοτήτων των μοντέλων και των agents (Jimenez et al., 2024).

Η εξέλιξη των AI agents δείχνει τελικά ότι η Τεχνητή Νοημοσύνη αρχίζει να μετακινείται από τη θέση ενός εργαλείου που απλώς προτείνει κώδικα προς συστήματα που μπορούν να πραγματοποιούν μέρος της ίδιας της διαδικασίας ανάπτυξης. Η δυνατότητα περιήγησης σε αποθετήρια, επεξεργασίας αρχείων και εκτέλεσης δοκιμών, όπως παρουσιάζεται στο SWE-agent, αποτελεί χαρακτηριστικό παράδειγμα αυτής της μεταβολής. Πρόκειται για μία σημαντική εξέλιξη, επειδή αυξάνει το εύρος των εργασιών που μπορούν να ανατεθούν στην Τεχνητή Νοημοσύνη (Yang et al., 2024).

5.3 Μετασχηματισμός του ρόλου και των δεξιοτήτων του προγραμματιστή

Η αυξανόμενη ενσωμάτωση της Τεχνητής Νοημοσύνης στη μηχανική λογισμικού αναμένεται να επηρεάσει σημαντικά τον ρόλο του προγραμματιστή. Η αλλαγή αυτή δεν αφορά μόνο την εισαγωγή νέων εργαλείων στην καθημερινή εργασία, αλλά και τη σταδιακή μεταβολή των καθηκόντων που εκτελεί ο ίδιος ο επαγγελματίας. Καθώς τα εργαλεία AI μπορούν να παράγουν κώδικα, να προτείνουν διορθώσεις, να δημιουργούν δοκιμές και να υποστηρίζουν την επίλυση τεχνικών προβλημάτων, ένα μέρος της εργασίας μετακινείται από τη χειροκίνητη συγγραφή κώδικα προς την καθοδήγηση, τον έλεγχο και την αξιολόγηση των αποτελεσμάτων που παράγονται με τη βοήθεια της Τεχνητής Νοημοσύνης (Vaithilingam et al., 2022).

Η μεταβολή αυτή γίνεται περισσότερο εμφανής με την ανάπτυξη των AI agents που εξετάστηκαν στην προηγούμενη ενότητα. Όσο ένα σύστημα αποκτά τη δυνατότητα να αναλύει ένα πρόβλημα, να επεξεργάζεται αρχεία και να πραγματοποιεί διαδοχικές ενέργειες, τόσο μεγαλύτερο μέρος της τεχνικής εκτέλεσης μπορεί να αυτοματοποιηθεί. Αυτό σημαίνει ότι ο προγραμματιστής ενδέχεται να μην χρειάζεται να πραγματοποιεί ο ίδιος κάθε επιμέρους βήμα, αλλά να καθορίζει περισσότερο τον στόχο της εργασίας, να παρακολουθεί την εκτέλεσή της και να ελέγχει εάν το αποτέλεσμα είναι κατάλληλο για το συγκεκριμένο έργο (Liu et al., 2024).

Έτσι, το βασικό ερώτημα για το μέλλον δεν περιορίζεται στο εάν η Τεχνητή Νοημοσύνη θα «αντικαταστήσει τους προγραμματιστές». Περισσότερο χρήσιμο είναι να εξεταστεί ποια καθήκοντα των προγραμματιστών μπορούν να αυτοματοποιηθούν και ποια εξακολουθούν να απαιτούν ανθρώπινη γνώση και κρίση. Η παραγωγή τυποποιημένου κώδικα, ορισμένες μορφές τεκμηρίωσης, η δημιουργία απλών δοκιμών και κάποιες επαναλαμβανόμενες διορθώσεις αποτελούν παραδείγματα εργασιών στις οποίες η AI μπορεί να αναλάβει μεγαλύτερο μέρος της εκτέλεσης. Αντίθετα, η κατανόηση σύνθετων απαιτήσεων και η λήψη αποφάσεων που επηρεάζουν συνολικά ένα πληροφοριακό σύστημα παραμένουν περισσότερο σύνθετες δραστηριότητες (Liu et al., 2024).

Η εξέλιξη αυτή μπορεί να περιορίσει τον χρόνο που αφιερώνει ο προγραμματιστής στη χειροκίνητη παραγωγή κώδικα, χωρίς όμως να καταργεί τη σημασία της γνώσης προγραμματισμού. Για να αξιολογήσει μία λύση που παράγεται από AI, ο επαγγελματίας πρέπει να κατανοεί τη γλώσσα προγραμματισμού, τη λειτουργία του κώδικα και τις πιθανές συνέπειες μιας αλλαγής. Διαφορετικά, υπάρχει κίνδυνος να ενσωματώνει προτάσεις χωρίς να μπορεί να αναγνωρίσει λογικά σφάλματα, προβλήματα απόδοσης ή ευπάθειες ασφαλείας. Η αξιολόγηση

του παραγόμενου αποτελέσματος αποτελεί επομένως ουσιαστικό μέρος της αποτελεσματικής χρήσης των εργαλείων AI (Vaithilingam et al., 2022).

Ιδιαίτερη σημασία αναμένεται να αποκτήσει η ικανότητα του προγραμματιστή να διατυπώνει με σαφήνεια το πρόβλημα και τις απαιτήσεις του. Όταν η AI χρησιμοποιείται για την παραγωγή ή τροποποίηση κώδικα, η ποιότητα του αποτελέσματος εξαρτάται σε σημαντικό βαθμό από τις πληροφορίες και το πλαίσιο που παρέχονται στο σύστημα. Μία ασαφής περιγραφή μπορεί να οδηγήσει σε μία τεχνικά σωστή αλλά ακατάλληλη λύση. Συνεπώς, η ικανότητα μετατροπής των αναγκών ενός έργου σε σαφείς τεχνικές οδηγίες αποκτά μεγαλύτερη αξία σε ένα περιβάλλον όπου μέρος της υλοποίησης πραγματοποιείται με τη βοήθεια της Τεχνητής Νοημοσύνης (Vaithilingam et al., 2022).

Παράλληλα, η κριτική σκέψη αναμένεται να αποτελέσει ακόμη σημαντικότερη δεξιότητα. Η παραγωγή μιας πειστικής απάντησης από ένα γλωσσικό μοντέλο δεν αποτελεί απόδειξη ότι η λύση είναι σωστή. Ο προγραμματιστής πρέπει να μπορεί να αμφισβητεί το αποτέλεσμα, να εξετάζει εναλλακτικές λύσεις και να αναγνωρίζει περιπτώσεις στις οποίες η πρόταση του συστήματος δεν ανταποκρίνεται στις πραγματικές απαιτήσεις. Η ανάγκη αυτή γίνεται μεγαλύτερη όσο περισσότερο οι προτάσεις της AI ενσωματώνονται στην καθημερινή διαδικασία ανάπτυξης λογισμικού (Vaithilingam et al., 2022).

Αντίστοιχα, μεγαλύτερη βαρύτητα μπορεί να αποκτήσει η αρχιτεκτονική σκέψη. Η Τεχνητή Νοημοσύνη μπορεί να διευκολύνει την παραγωγή επιμέρους τμημάτων κώδικα, αλλά η ανάπτυξη ενός ποιοτικού συστήματος απαιτεί συνολικές αποφάσεις σχετικά με τη δομή, τις εξαρτήσεις, την επεκτασιμότητα και τη συντηρησιμότητά του. Ένα απόσπασμα κώδικα μπορεί να επιλύει σωστά ένα μεμονωμένο πρόβλημα χωρίς να αποτελεί την καλύτερη επιλογή για τη συνολική αρχιτεκτονική. Η δυνατότητα κατανόησης αυτών των σχέσεων παραμένει κρίσιμη για τη μηχανική λογισμικού, ιδιαίτερα σε μεγάλα και σύνθετα έργα (Pressman & Maxim, 2020).

Ανάλογη σημασία αποκτούν οι δεξιότητες που σχετίζονται με τον έλεγχο και τις δοκιμές του λογισμικού. Όταν αυξάνεται η ποσότητα του κώδικα που μπορεί να δημιουργηθεί αυτόματα, αυξάνεται παράλληλα η ανάγκη να επιβεβαιώνεται ότι ο κώδικας λειτουργεί σωστά και δεν προκαλεί προβλήματα σε άλλα μέρη της εφαρμογής. Η δημιουργία κατάλληλων δοκιμών, η αξιολόγηση των αποτελεσμάτων τους και η αναγνώριση περιπτώσεων που δεν έχουν εξεταστεί επαρκώς αποτελούν συνεπώς δεξιότητες που διατηρούν τη σημασία τους ακόμη και σε ένα περιβάλλον αυξημένης αυτοματοποίησης (Pressman & Maxim, 2020).

Η εξέλιξη της AI είναι πιθανό να επηρεάσει διαφορετικά και τους νέους ή λιγότερο έμπειρους προγραμματιστές. Τα εργαλεία παραγωγής κώδικα μπορούν να λειτουργήσουν ως σημαντικό μέσο υποστήριξης, καθώς επιτρέπουν την άμεση παροχή παραδειγμάτων, επεξηγήσεων και προτεινόμενων λύσεων. Ένας αρχάριος μπορεί να χρησιμοποιήσει την AI για να κατανοήσει μία άγνωστη συνάρτηση ή να ζητήσει επεξήγηση ενός σφάλματος. Παράλληλα, όμως, η ευκολία παραγωγής έτοιμου κώδικα δημιουργεί τον κίνδυνο ο χρήστης να αποδέχεται μία λύση χωρίς να κατανοεί επαρκώς τον τρόπο λειτουργίας της (Vaithilingam et al., 2022).

Το ζήτημα αυτό είναι σημαντικό και για την εκπαίδευση των μελλοντικών προγραμματιστών. Εάν ένα σημαντικό μέρος του κώδικα μπορεί να δημιουργείται αυτόματα, η εκπαίδευση δεν μπορεί να περιορίζεται αποκλειστικά στην εκμάθηση της σύνταξης μιας γλώσσας προγραμματισμού. Παραμένει απαραίτητη η γνώση των βασικών αρχών του προγραμματισμού, αλλά μεγαλύτερη έμφαση μπορεί να χρειαστεί να δοθεί στην επίλυση προβλημάτων, στους αλγορίθμους, στον σχεδιασμό συστημάτων, στην ασφάλεια και στην κριτική αξιολόγηση κώδικα που έχει παραχθεί από τρίτους ή από AI (Becker et al., 2023).

Παράλληλα, η χρήση της AI δημιουργεί την ανάγκη για μία νέα μορφή AI literacy στον χώρο της μηχανικής λογισμικού. Ο προγραμματιστής δεν αρκεί να γνωρίζει πώς να χρησιμοποιήσει ένα εργαλείο, αλλά χρειάζεται να κατανοεί σε βασικό επίπεδο τις δυνατότητες και τους περιορισμούς του. Πρέπει να γνωρίζει ότι ένα μεγάλο γλωσσικό μοντέλο μπορεί να δημιουργήσει λανθασμένες απαντήσεις, ότι το αποτέλεσμα απαιτεί έλεγχο και ότι η εισαγωγή εμπιστευτικών πληροφοριών σε μία εξωτερική υπηρεσία μπορεί να δημιουργήσει προβλήματα προστασίας δεδομένων. Η γνώση αυτών των περιορισμών αποτελεί μέρος της υπεύθυνης επαγγελματικής χρήσης της Τεχνητής Νοημοσύνης (Bender et al., 2021).

Η αυτοματοποίηση μπορεί παράλληλα να επηρεάσει τη σύνθεση των ομάδων ανάπτυξης λογισμικού. Εφόσον περισσότερες τυποποιημένες εργασίες μπορούν να υποστηρίζονται από AI, οι ομάδες μπορεί να οργανώνουν διαφορετικά την κατανομή των καθηκόντων τους. Η αξία ενός

επαγγελματία ενδέχεται να συνδέεται περισσότερο με την ικανότητά του να συνδυάζει τεχνική γνώση, κατανόηση των επιχειρησιακών αναγκών και αποτελεσματική αξιοποίηση των εργαλείων AI. Η ανάπτυξη agents που μπορούν να αναλαμβάνουν μεγαλύτερο μέρος μιας διαδικασίας ενισχύει αυτή την πιθανότητα μεταβολής του τρόπου οργάνωσης της εργασίας (Liu et al., 2024).

Αυτό δεν σημαίνει ότι όλες οι ειδικότητες ή όλα τα επίπεδα εμπειρίας θα επηρεαστούν με τον ίδιο τρόπο. Οι εργασίες που είναι περισσότερο επαναλαμβανόμενες και μπορούν να περιγραφούν με σαφείς κανόνες είναι πιθανότερο να αυτοματοποιηθούν σε μεγαλύτερο βαθμό. Αντίθετα, δραστηριότητες που απαιτούν κατανόηση ασαφών απαιτήσεων, επικοινωνία με πελάτες και χρήστες, αξιολόγηση διαφορετικών συμβιβασμών και ανάληψη ευθύνης για κρίσιμες αποφάσεις είναι δυσκολότερο να μεταφερθούν πλήρως σε αυτοματοποιημένα συστήματα. Η δυσκολία των μοντέλων στην αντιμετώπιση πραγματικών και σύνθετων προβλημάτων λογισμικού δείχνει ότι η ανθρώπινη συμμετοχή παραμένει ουσιαστική (Jimenez et al., 2024).

Επομένως, είναι περισσότερο ακριβές να εξετάζεται η Τεχνητή Νοημοσύνη ως παράγοντας μετασχηματισμού του επαγγέλματος και όχι να θεωρείται δεδομένη η πλήρης αντικατάσταση των προγραμματιστών. Ορισμένες εργασίες μπορούν να περιοριστούν, άλλες να αυτοματοποιηθούν σε μεγάλο βαθμό και παράλληλα να δημιουργηθούν νέα καθήκοντα που σχετίζονται με την επίβλεψη, την αξιολόγηση και τον έλεγχο των συστημάτων AI. Η ιστορία της μηχανικής λογισμικού άλλωστε χαρακτηρίζεται από συνεχή εμφάνιση εργαλείων που αυτοματοποιούν επιμέρους δραστηριότητες, χωρίς αυτό να εξαλείφει την ανάγκη για ανθρώπινη επίλυση προβλημάτων και λήψη αποφάσεων (Pressman & Maxim, 2020).

Η εξέλιξη αυτή απαιτεί και μία διαφορετική αντίληψη για τη συνεχή επαγγελματική εκπαίδευση. Οι γνώσεις που χρειάζεται ένας προγραμματιστής μεταβάλλονται καθώς εμφανίζονται νέα εργαλεία, πλατφόρμες και τρόποι ανάπτυξης. Η ικανότητα προσαρμογής στις τεχνολογικές αλλαγές ήταν πάντοτε σημαντική στη μηχανική λογισμικού, όμως η ταχύτητα εξέλιξης της Τεχνητής Νοημοσύνης ενισχύει ακόμη περισσότερο αυτή την ανάγκη. Οι επαγγελματίες χρειάζεται να παρακολουθούν τις νέες δυνατότητες, αλλά και να γνωρίζουν τους περιορισμούς και τους κινδύνους των εργαλείων που ενσωματώνουν στην εργασία τους (Bender et al., 2021).

Μελλοντικά, επομένως, ένας αποτελεσματικός μηχανικός λογισμικού είναι πιθανό να χρειάζεται έναν συνδυασμό παραδοσιακών και νέων δεξιοτήτων. Η γνώση προγραμματισμού, αλγορίθμων, βάσεων δεδομένων, αρχιτεκτονικής, testing και ασφάλειας παραμένει απαραίτητη, αλλά συμπληρώνεται από την ικανότητα αποτελεσματικής συνεργασίας με συστήματα AI. Ο προγραμματιστής θα πρέπει να μπορεί να καθορίζει σωστά το πρόβλημα, να παρέχει κατάλληλο πλαίσιο στο σύστημα, να αξιολογεί τις προτάσεις του και να γνωρίζει πότε μία εργασία δεν πρέπει να ανατεθεί αποκλειστικά στην Τεχνητή Νοημοσύνη (Liu et al., 2024).

5.4 Συνεργασία ανθρώπου και Τεχνητής Νοημοσύνης στη δημιουργία λογισμικού

Η εξέλιξη της Τεχνητής Νοημοσύνης στη μηχανική λογισμικού οδηγεί σταδιακά στη διαμόρφωση ενός νέου μοντέλου εργασίας, στο οποίο ο άνθρωπος και τα συστήματα AI δεν λειτουργούν απαραίτητα ανταγωνιστικά, αλλά μπορούν να αναλαμβάνουν διαφορετικούς και συμπληρωματικούς ρόλους. Τα εργαλεία παραγωγής κώδικα και οι AI agents μπορούν να αναλάβουν μέρος των επαναλαμβανόμενων ή περισσότερο τυποποιημένων εργασιών, ενώ ο προγραμματιστής εξακολουθεί να έχει καθοριστικό ρόλο στον προσδιορισμό των απαιτήσεων, στην αξιολόγηση των λύσεων και στη λήψη κρίσιμων αποφάσεων. Η συνεργασία ανθρώπου και AI αποτελεί επομένως μία από τις βασικές κατευθύνσεις προς τις οποίες μπορεί να εξελιχθεί η ανάπτυξη λογισμικού τα επόμενα χρόνια (Liu et al., 2024).

Η συγκεκριμένη συνεργασία διαφοροποιείται από την απλή χρήση ενός εργαλείου αυτοματοποίησης. Σε ένα παραδοσιακό σύστημα, ο προγραμματιστής χρησιμοποιεί ένα εργαλείο για να εκτελέσει μία σαφώς προσδιορισμένη λειτουργία. Αντίθετα, τα σύγχρονα συστήματα Τεχνητής Νοημοσύνης μπορούν να προτείνουν εναλλακτικές λύσεις, να παράγουν κώδικα και να υποστηρίζουν τον χρήστη κατά την επίλυση ενός προβλήματος. Ο άνθρωπος μπορεί να αξιολογήσει το αποτέλεσμα, να δώσει πρόσθετες οδηγίες και να ζητήσει τροποποιήσεις, δημιουργώντας έτσι μία περισσότερο διαδραστική και επαναληπτική διαδικασία ανάπτυξης (Vaithilingam et al., 2022).

Η μορφή αυτή συνεργασίας μπορεί να γίνει κατανοητή ως μία διαδικασία κατανομής εργασιών ανάμεσα στον άνθρωπο και την AI. Η Τεχνητή Νοημοσύνη είναι ιδιαίτερα χρήσιμη σε εργασίες που απαιτούν γρήγορη επεξεργασία πληροφοριών, δημιουργία εναλλακτικών προτάσεων ή παραγωγή αρχικού κώδικα. Αντίθετα, ο άνθρωπος έχει σημαντικό ρόλο όταν απαιτείται κατανόηση του ευρύτερου πλαισίου, αξιολόγηση διαφορετικών τεχνικών επιλογών και λήψη αποφάσεων που επηρεάζουν το σύνολο του έργου. Η αποτελεσματικότητα της συνεργασίας εξαρτάται επομένως από την κατάλληλη κατανομή των εργασιών και όχι από την προσπάθεια πλήρους μεταφοράς της ανάπτυξης σε ένα αυτοματοποιημένο σύστημα (Liu et al., 2024).

Η συνεργασία αυτή μπορεί να εφαρμοστεί και πέρα από την αρχική συγγραφή του κώδικα. Ένα σύστημα AI μπορεί να χρησιμοποιηθεί για την επεξήγηση ενός υπάρχοντος προγράμματος, τη δημιουργία τεκμηρίωσης, την πρόταση περιπτώσεων δοκιμών ή την αναζήτηση πιθανών σφαλμάτων. Ο προγραμματιστής μπορεί στη συνέχεια να αξιολογήσει εάν οι προτάσεις ανταποκρίνονται στις πραγματικές απαιτήσεις και να αποφασίσει ποιες από αυτές θα χρησιμοποιηθούν. Η δυνατότητα αυτή είναι ιδιαίτερα σημαντική σε μεγάλα έργα, όπου σημαντικό μέρος του χρόνου των μηχανικών λογισμικού αφιερώνεται στην κατανόηση και τροποποίηση ήδη υπάρχοντος κώδικα και όχι αποκλειστικά στη δημιουργία νέου (Pressman & Maxim, 2020).

Η έννοια του human-in-the-loop αποκτά ιδιαίτερη σημασία στο πλαίσιο αυτό. Με τον όρο περιγράφεται μία προσέγγιση κατά την οποία ο άνθρωπος παραμένει ενεργό μέρος της διαδικασίας και παρεμβαίνει σε κατάλληλα σημεία για να αξιολογήσει, να διορθώσει ή να εγκρίνει το αποτέλεσμα του αυτοματοποιημένου συστήματος. Στη μηχανική λογισμικού, αυτό μπορεί να σημαίνει ότι η AI δημιουργεί μία προτεινόμενη αλλαγή, αλλά ο προγραμματιστής την εξετάζει πριν αυτή ενσωματωθεί στον κύριο κώδικα ή διατεθεί σε πραγματικό περιβάλλον λειτουργίας. Η προσέγγιση αυτή αποκτά μεγαλύτερη σημασία όσο αυξάνονται οι δυνατότητες των AI agents να πραγματοποιούν ενέργειες με μεγαλύτερο βαθμό αυτονομίας (Liu et al., 2024).

Η ανθρώπινη συμμετοχή είναι απαραίτητη κυρίως επειδή τα συστήματα Τεχνητής Νοημοσύνης δεν παρέχουν εγγύηση ορθότητας για κάθε αποτέλεσμα που παράγουν. Ένα μοντέλο μπορεί να δημιουργήσει κώδικα που φαίνεται σωστός και εκτελείται χωρίς άμεσο σφάλμα, αλλά δεν ανταποκρίνεται πλήρως στις απαιτήσεις του έργου ή περιλαμβάνει προβλήματα που εμφανίζονται μόνο σε συγκεκριμένες συνθήκες. Η αξιολόγηση της χρησιμότητας των εργαλείων παραγωγής κώδικα έχει δείξει ότι η κατανόηση και η επαλήθευση των προτάσεων αποτελούν σημαντικό μέρος της εμπειρίας του προγραμματιστή κατά τη χρήση τους (Vaithilingam et al., 2022).

Η συνεργασία ανθρώπου και AI μπορεί επίσης να λειτουργήσει ως μορφή AI-assisted pair programming. Στο παραδοσιακό pair programming, δύο προγραμματιστές συνεργάζονται για την ανάπτυξη και τον έλεγχο του κώδικα. Στη νέα αυτή μορφή, ένα σύστημα AI μπορεί να λειτουργεί ως ψηφιακός βοηθός, προτείνοντας κώδικα, πιθανές λύσεις ή επεξηγήσεις, ενώ ο άνθρωπος διατηρεί την τελική κρίση σχετικά με το αποτέλεσμα. Παρότι η σχέση αυτή δεν είναι ισοδύναμη με τη συνεργασία δύο ανθρώπων, παρουσιάζει ορισμένα κοινά στοιχεία, καθώς ο προγραμματιστής δεν εργάζεται πλέον αποκλειστικά μόνος του κατά την παραγωγή μιας λύσης (Vaithilingam et al., 2022).

Η συνεργασία αυτή μπορεί να προσφέρει ιδιαίτερα οφέλη όταν ο προγραμματιστής βρίσκεται αντιμέτωπος με άγνωστο ή σύνθετο κώδικα. Η AI μπορεί να χρησιμοποιηθεί για μία πρώτη επεξήγηση της λειτουργίας μιας συνάρτησης ή για τον εντοπισμό πιθανών σχέσεων μεταξύ διαφορετικών τμημάτων του συστήματος. Ωστόσο, η τελική κατανόηση δεν πρέπει να βασίζεται αποκλειστικά στην απάντηση του μοντέλου, καθώς η επεξήγηση μπορεί να είναι ελλιπής ή λανθασμένη. Η αποτελεσματική συνεργασία προϋποθέτει επομένως ότι ο άνθρωπος χρησιμοποιεί το σύστημα για να επιταχύνει την κατανόηση, χωρίς να εγκαταλείπει τη δική του διαδικασία επαλήθευσης (Vaithilingam et al., 2022).

Η ανθρώπινη επίβλεψη γίνεται ακόμη σημαντικότερη στην περίπτωση των AI agents. Ένας agent μπορεί να έχει τη δυνατότητα να επεξεργάζεται πολλά αρχεία, να εκτελεί εντολές και να πραγματοποιεί διαδοχικές αλλαγές. Όσο μεγαλύτερη είναι η αυτονομία του συστήματος, τόσο μεγαλύτερες μπορεί να είναι και οι συνέπειες μιας λανθασμένης απόφασης. Η έρευνα πάνω σε συστήματα όπως το SWE-agent αναδεικνύει τη σημασία της σχεδίασης της αλληλεπίδρασης μεταξύ του agent και του υπολογιστικού περιβάλλοντος στο οποίο λειτουργεί (Yang et al., 2024).

Για τον λόγο αυτό, ένα πιθανό μελλοντικό μοντέλο ανάπτυξης είναι η ύπαρξη διαφορετικών επιπέδων ανθρώπινης έγκρισης. Απλές και χαμηλού κινδύνου εργασίες θα μπορούσαν να πραγματοποιούνται περισσότερο αυτόνομα, ενώ αλλαγές που αφορούν κρίσιμα μέρη μιας εφαρμογής θα απαιτούν ανθρώπινη επιβεβαίωση. Ένας agent, για παράδειγμα, μπορεί να προτείνει μία σειρά αλλαγών και να πραγματοποιήσει τις απαραίτητες δοκιμές, αλλά η τελική ενσωμάτωση στον κύριο κώδικα να πραγματοποιείται μόνο μετά από ανθρώπινο code review. Η προσέγγιση αυτή επιτρέπει την αξιοποίηση της αυτοματοποίησης χωρίς να καταργεί τον ανθρώπινο έλεγχο (Liu et al., 2024).

Παράλληλα, η αποτελεσματική συνεργασία προϋποθέτει ότι ο άνθρωπος μπορεί να κατανοεί τι έχει κάνει το σύστημα. Εάν ένας agent πραγματοποιήσει πολλές αλλαγές χωρίς σαφή καταγραφή των ενεργειών και της λογικής που ακολούθησε, ο έλεγχος του αποτελέσματος γίνεται δυσκολότερος. Για τον λόγο αυτό, η διαφάνεια, η καταγραφή των αλλαγών και η δυνατότητα αναθεώρησης των ενεργειών αποτελούν σημαντικά χαρακτηριστικά για τη δημιουργία περισσότερο αξιόπιστων συστημάτων υποστήριξης της ανάπτυξης λογισμικού (Liu et al., 2024).

Η συνεργασία ανθρώπου και AI δεν αφορά όμως μόνο τον έλεγχο των λαθών της Τεχνητής Νοημοσύνης. Μπορεί να δημιουργήσει μία διαδικασία στην οποία οι δυνατότητες των δύο πλευρών συμπληρώνουν η μία την άλλη. Η AI μπορεί να επεξεργάζεται γρήγορα μεγάλο όγκο πληροφοριών και να δημιουργεί πολλές πιθανές λύσεις, ενώ ο άνθρωπος μπορεί να χρησιμοποιεί την εμπειρία και την κατανόηση του ευρύτερου πλαισίου για να επιλέγει ποια λύση είναι περισσότερο κατάλληλη. Η αξία της συνεργασίας βρίσκεται επομένως στον συνδυασμό της ταχύτητας της αυτοματοποίησης με την ανθρώπινη κρίση και όχι απλώς στη μεταφορά όσο το δυνατόν περισσότερων εργασιών σε ένα μοντέλο (Liu et al., 2024).

Η προσέγγιση αυτή μπορεί να επηρεάσει και τον τρόπο με τον οποίο οργανώνονται οι ομάδες ανάπτυξης. Σε ένα περιβάλλον όπου οι μηχανικοί χρησιμοποιούν συστηματικά εργαλεία AI, η συνεργασία δεν περιορίζεται πλέον αποκλειστικά στις σχέσεις μεταξύ των μελών της ομάδας, αλλά περιλαμβάνει και την αποτελεσματική αξιοποίηση των αυτοματοποιημένων συστημάτων. Οι ομάδες χρειάζεται επομένως να καθορίζουν ποια εργαλεία μπορούν να χρησιμοποιούνται, σε ποιες εργασίες, ποια δεδομένα επιτρέπεται να εισάγονται και ποιος είναι υπεύθυνος για τον έλεγχο του αποτελέσματος. Η ανάπτυξη των LLM-based agents καθιστά τέτοιους οργανωτικούς μηχανισμούς περισσότερο σημαντικούς όσο αυξάνεται η αυτονομία των συστημάτων (Liu et al., 2024).

Παρά την τεχνολογική πρόοδο, υπάρχουν στοιχεία της ανάπτυξης λογισμικού στα οποία η ανθρώπινη επικοινωνία εξακολουθεί να έχει ιδιαίτερη σημασία. Η κατανόηση των αναγκών ενός πελάτη, η διαπραγμάτευση αντικρουόμενων απαιτήσεων και η συνεργασία μεταξύ διαφορετικών επαγγελματικών ομάδων δεν αποτελούν απλώς προβλήματα παραγωγής κώδικα. Η μηχανική λογισμικού περιλαμβάνει τεχνικές αλλά και οργανωτικές και ανθρώπινες διαστάσεις, γεγονός που καθιστά δύσκολη την πλήρη μεταφορά της διαδικασίας σε αυτόνομα συστήματα (Pressman & Maxim, 2020).

Η συνεργασία ανθρώπου και Τεχνητής Νοημοσύνης δημιουργεί επομένως ένα μοντέλο στο οποίο η αυτοματοποίηση και η ανθρώπινη ευθύνη πρέπει να συνυπάρχουν. Όσο περισσότερο το σύστημα συμμετέχει στην παραγωγή λογισμικού, τόσο σημαντικότερο γίνεται να είναι σαφές ποιος ελέγχει το αποτέλεσμα και ποιος λαμβάνει την τελική απόφαση για την ενσωμάτωσή του. Η AI μπορεί να υποστηρίζει την τεχνική εργασία, αλλά η επαγγελματική ευθύνη για ένα σύστημα που διατίθεται σε πραγματικούς χρήστες δεν μπορεί να θεωρείται ότι μεταφέρεται αυτόματα στο εργαλείο που συνέβαλε στη δημιουργία του (Pearce et al., 2022).

5.5 Προοπτικές και προϋποθέσεις για την υπεύθυνη αξιοποίηση της Τεχνητής Νοημοσύνης στη μηχανική λογισμικού

Η μελλοντική αξιοποίηση της Τεχνητής Νοημοσύνης στη μηχανική λογισμικού δεν εξαρτάται μόνο από το πόσο εξελιγμένα θα γίνουν τα μοντέλα ή από το πόσες εργασίες θα μπορούν να αυτοματοποιήσουν. Εξίσου σημαντικός είναι ο τρόπος με τον οποίο οι τεχνολογίες αυτές θα ενσωματωθούν στην πραγματική διαδικασία ανάπτυξης. Η δυνατότητα παραγωγής κώδικα, χρήσης εξωτερικών εργαλείων και εκτέλεσης περισσότερο σύνθετων εργασιών δημιουργεί σημαντικές προοπτικές, αλλά παράλληλα αυξάνει την ανάγκη για έλεγχο, αξιολόγηση και σαφείς

κανόνες χρήσης. Η υπεύθυνη αξιοποίηση της AI προϋποθέτει επομένως μία ισορροπία ανάμεσα στην αυτοματοποίηση και στην ανθρώπινη εποπτεία (Liu et al., 2024).

Μία από τις βασικότερες προϋποθέσεις είναι η διατήρηση του ανθρώπινου ελέγχου στις κρίσιμες αποφάσεις. Η Τεχνητή Νοημοσύνη μπορεί να παράγει προτάσεις ή ακόμη και να πραγματοποιεί συγκεκριμένες ενέργειες, όμως αυτό δεν σημαίνει ότι κάθε αποτέλεσμα πρέπει να ενσωματώνεται αυτόματα σε ένα πραγματικό έργο. Όσο μεγαλύτερος είναι ο πιθανός αντίκτυπος μιας αλλαγής, τόσο σημαντικότερη γίνεται η ανθρώπινη αξιολόγηση πριν από την εφαρμογή της. Η ανάγκη αυτή ενισχύεται από το γεγονός ότι ακόμη και προηγμένα συστήματα αντιμετωπίζουν δυσκολίες σε σύνθετες εργασίες που απαιτούν κατανόηση πραγματικών αποθετηρίων λογισμικού (Jimenez et al., 2024).

Η ανθρώπινη εποπτεία δεν πρέπει, ωστόσο, να περιορίζεται σε έναν τυπικό έλεγχο του παραγόμενου κώδικα. Ο προγραμματιστής χρειάζεται να κατανοεί τι έχει δημιουργήσει το σύστημα, γιατί μία συγκεκριμένη λύση θεωρείται κατάλληλη και ποιες μπορεί να είναι οι επιπτώσεις της στο υπόλοιπο έργο. Η χρήση ενός AI-generated αποτελέσματος χωρίς επαρκή κατανόηση μπορεί να δημιουργήσει προβλήματα που δεν είναι άμεσα εμφανή. Για τον λόγο αυτό, η ικανότητα αξιολόγησης των προτάσεων της AI αποτελεί ουσιαστικό στοιχείο της υπεύθυνης ενσωμάτωσής της στη μηχανική λογισμικού (Vaithilingam et al., 2022).

Η απαίτηση αυτή γίνεται ακόμη πιο σημαντική όταν χρησιμοποιούνται AI agents με μεγαλύτερο βαθμό αυτονομίας. Ένα σύστημα που μπορεί να αλλάζει πολλά αρχεία, να εκτελεί εντολές και να πραγματοποιεί διαδοχικές ενέργειες έχει τη δυνατότητα να επηρεάσει μεγαλύτερο μέρος ενός έργου από ένα εργαλείο που απλώς προτείνει μία γραμμή κώδικα. Η ανάπτυξη agent-based συστημάτων στη μηχανική λογισμικού δημιουργεί συνεπώς την ανάγκη για μηχανισμούς που θα περιορίζουν τις επιτρεπόμενες ενέργειες και θα επιτρέπουν την ανθρώπινη παρέμβαση όταν απαιτείται (Liu et al., 2024).

Στο πλαίσιο αυτό, ιδιαίτερη σημασία έχει η εφαρμογή της αρχής της ελάχιστης απαραίτητης πρόσβασης. Ένα σύστημα Τεχνητής Νοημοσύνης θα πρέπει να έχει πρόσβαση μόνο στα δεδομένα, στα αρχεία και στις λειτουργίες που είναι απαραίτητα για την εκτέλεση της συγκεκριμένης εργασίας. Η παροχή υπερβολικά διευρυμένων δικαιωμάτων σε έναν agent μπορεί να αυξήσει τις συνέπειες ενός σφάλματος ή μιας ανεπιθύμητης ενέργειας. Καθώς οι agents συνδέουν γλωσσικά μοντέλα με εργαλεία και υπολογιστικά περιβάλλοντα, ο έλεγχος των δυνατοτήτων δράσης τους αποτελεί βασικό μέρος της ασφαλούς σχεδίασής τους (Liu et al., 2024).

Η υπεύθυνη αξιοποίηση προϋποθέτει επίσης μεγαλύτερη διαφάνεια ως προς τη χρήση της AI μέσα σε ένα έργο. Όταν σημαντικά τμήματα κώδικα δημιουργούνται ή τροποποιούνται από αυτοματοποιημένα συστήματα, είναι χρήσιμο να υπάρχει η δυνατότητα αναγνώρισης και ελέγχου αυτών των παρεμβάσεων. Η καταγραφή των αλλαγών αποκτά ακόμη μεγαλύτερη σημασία στην περίπτωση των agents, επειδή ένα σύστημα μπορεί να πραγματοποιήσει πολλές διαδοχικές ενέργειες για την επίτευξη ενός στόχου. Η δυνατότητα αναθεώρησης αυτών των ενεργειών μπορεί να βοηθήσει τόσο στην αξιολόγηση του αποτελέσματος όσο και στον εντοπισμό της αιτίας ενός πιθανού προβλήματος (Liu et al., 2024).

Η υπεύθυνη αξιοποίηση της AI απαιτεί παράλληλα την ανάπτυξη οργανωτικών πολιτικών χρήσης. Μία επιχείρηση ή μία ομάδα ανάπτυξης χρειάζεται να γνωρίζει ποια εργαλεία επιτρέπεται να χρησιμοποιούνται, για ποιες κατηγορίες εργασιών, ποια δεδομένα μπορούν να παρέχονται σε αυτά και ποια επίπεδα ελέγχου απαιτούνται πριν από την ενσωμάτωση του παραγόμενου αποτελέσματος. Η ατομική κρίση κάθε προγραμματιστή παραμένει σημαντική, αλλά δεν είναι επαρκής όταν η AI χρησιμοποιείται συστηματικά σε μεγάλα έργα και από πολλές διαφορετικές ομάδες (Liu et al., 2024).

Παράλληλα, η αξιολόγηση των εργαλείων AI πρέπει να πραγματοποιείται με βάση ρεαλιστικά κριτήρια και πραγματικές εργασίες μηχανικής λογισμικού. Η υψηλή απόδοση ενός μοντέλου σε μία περιορισμένη δοκιμή παραγωγής κώδικα δεν σημαίνει ότι μπορεί να αντιμετωπίσει με την ίδια επιτυχία ένα μεγάλο έργο. Το SWE-bench ανέδειξε ακριβώς αυτή τη διαφορά, καθώς η επίλυση πραγματικών ζητημάτων από υπάρχοντα αποθετήρια απαιτεί κατανόηση μεγαλύτερου πλαισίου, εντοπισμό των κατάλληλων αρχείων και παραγωγή αλλαγών που πρέπει να λειτουργούν μέσα σε ένα ήδη υπάρχον σύστημα (Jimenez et al., 2024).

Για τον λόγο αυτό, η μελλοντική υιοθέτηση των εργαλείων δεν πρέπει να καθορίζεται αποκλειστικά από το πόσο εντυπωσιακές είναι οι δυνατότητές τους ή από το πόσο γρήγορα μπορούν να παράγουν κώδικα. Χρειάζεται να εξετάζεται κατά πόσο προσφέρουν πραγματική

βελτίωση στην παραγωγικότητα, χωρίς παράλληλη υποβάθμιση της ποιότητας, της ασφάλειας ή της δυνατότητας συντήρησης του λογισμικού. Η αξιολόγηση της Τεχνητής Νοημοσύνης στη μηχανική λογισμικού πρέπει επομένως να λαμβάνει υπόψη το συνολικό αποτέλεσμα της διαδικασίας και όχι μόνο την ταχύτητα παραγωγής ενός αρχικού κώδικα (Vaithilingam et al., 2022).

Μία ακόμη σημαντική προϋπόθεση είναι η αποφυγή της αντίληψης ότι μεγαλύτερη αυτονομία σημαίνει πάντοτε καλύτερο αποτέλεσμα. Σε ορισμένες εργασίες μπορεί να είναι αποτελεσματικό ένα σύστημα να πραγματοποιεί πολλές ενέργειες χωρίς συνεχή ανθρώπινη παρέμβαση. Σε άλλες, ιδιαίτερα όταν το λογισμικό είναι κρίσιμο ή επεξεργάζεται ευαίσθητα δεδομένα, μπορεί να είναι προτιμότερο να υπάρχουν περισσότερα σημεία ανθρώπινης έγκρισης. Η ανάπτυξη των AI agents δημιουργεί συνεπώς την ανάγκη για διαφορετικά επίπεδα αυτονομίας ανάλογα με τη φύση και τον κίνδυνο της εκάστοτε εργασίας (Liu et al., 2024).

Στην πράξη, ένα τέτοιο μοντέλο θα μπορούσε να σημαίνει ότι ένας agent αναλαμβάνει αυτόνομα εργασίες χαμηλότερου κινδύνου, ενώ απαιτείται έγκριση για σημαντικές αλλαγές στον κώδικα ή για ενέργειες που επηρεάζουν πραγματικά δεδομένα και συστήματα. Με αυτόν τον τρόπο, η αυτονομία μπορεί να αυξάνεται εκεί όπου προσφέρει πραγματικό όφελος, χωρίς να εξαλείφονται οι απαραίτητοι μηχανισμοί ελέγχου. Η δυνατότητα των agents να αλληλεπιδρούν με πραγματικά περιβάλλοντα ανάπτυξης καθιστά ιδιαίτερα σημαντικό τον σχεδιασμό των ορίων μέσα στα οποία επιτρέπεται να ενεργούν (Yang et al., 2024).

Η παράμετρος αυτή έχει σημασία και στη μηχανική λογισμικού, όπου τα εργαλεία AI μπορούν να χρησιμοποιούνται καθημερινά από μεγάλο αριθμό προγραμματιστών. Δεν είναι απαραίτητο κάθε εργασία να εκτελείται από το μεγαλύτερο και περισσότερο απαιτητικό μοντέλο, εφόσον ένα μικρότερο σύστημα μπορεί να προσφέρει επαρκές αποτέλεσμα. Η επιλογή κατάλληλων μοντέλων και η αποδοτική χρήση των υπολογιστικών πόρων μπορούν επομένως να αποτελέσουν μέρος μιας περισσότερο υπεύθυνης και βιώσιμης προσέγγισης στην αξιοποίηση της Τεχνητής Νοημοσύνης (Schwartz et al., 2020).

Παράλληλα, χρειάζεται να αναγνωριστεί ότι η τεχνολογία θα συνεχίσει να μεταβάλλεται με γρήγορο ρυθμό. Οι σημερινοί περιορισμοί των μοντέλων μπορεί να μειωθούν, ενώ παράλληλα μπορεί να εμφανιστούν νέοι κίνδυνοι που δεν είναι ακόμη πλήρως ορατοί. Για τον λόγο αυτό, οι πολιτικές και οι διαδικασίες που εφαρμόζονται σήμερα δεν πρέπει να θεωρούνται οριστικές. Η υπεύθυνη χρήση απαιτεί συνεχή αξιολόγηση των νέων εργαλείων και προσαρμογή των διαδικασιών ελέγχου στις πραγματικές δυνατότητες και αδυναμίες τους (Bender et al., 2021).

Η μελλοντική ανάπτυξη της Τεχνητής Νοημοσύνης στη μηχανική λογισμικού θα πρέπει επομένως να βασιστεί στην αρχή ότι η τεχνολογική δυνατότητα δεν ταυτίζεται αυτόματα με την κατάλληλη χρήση. Το γεγονός ότι ένα σύστημα μπορεί να δημιουργήσει ή να τροποποιήσει κώδικα χωρίς ανθρώπινη παρέμβαση δεν σημαίνει ότι αυτό είναι επιθυμητό σε κάθε περίπτωση. Η επιλογή του κατάλληλου βαθμού αυτοματοποίησης πρέπει να εξαρτάται από την πολυπλοκότητα της εργασίας, τις πιθανές συνέπειες ενός σφάλματος και την ικανότητα επαρκούς ελέγχου του αποτελέσματος (Liu et al., 2024).

Συμπεράσματα

Η Τεχνητή Νοημοσύνη αποτελεί πλέον μία από τις σημαντικότερες τεχνολογικές εξελίξεις που επηρεάζουν τη σύγχρονη ανάπτυξη λογισμικού. Η παρούσα εργασία εξέτασε τον τρόπο με τον οποίο η AI ενσωματώνεται στα διαφορετικά στάδια της μηχανικής λογισμικού, τις δυνατότητες που δημιουργεί για τους προγραμματιστές, αλλά και τους περιορισμούς και τους προβληματισμούς που συνοδεύουν την αυξανόμενη χρήση της. Από τη βιβλιογραφική μελέτη προκύπτει ότι η Τεχνητή Νοημοσύνη δεν αποτελεί πλέον μόνο ένα συμπληρωματικό εργαλείο, αλλά εξελίσσεται σταδιακά σε σημαντικό μέρος της διαδικασίας δημιουργίας λογισμικού (Liu et al., 2024).

Η εξέλιξη των μεγάλων γλωσσικών μοντέλων έχει συμβάλει σημαντικά σε αυτή τη μεταβολή. Τα σύγχρονα μοντέλα μπορούν να επεξεργάζονται φυσική γλώσσα και προγραμματιστικό κώδικα, επιτρέποντας στους προγραμματιστές να περιγράφουν ένα πρόβλημα ή μία επιθυμητή λειτουργία και να λαμβάνουν αυτόματα προτάσεις υλοποίησης. Η δυνατότητα αυτή έχει επεκτείνει σημαντικά το πεδίο εφαρμογής της Τεχνητής Νοημοσύνης στον προγραμματισμό, καθώς η χρήση της δεν περιορίζεται πλέον στην απλή αυτόματη συμπλήρωση κώδικα, αλλά περιλαμβάνει

παραγωγή νέου κώδικα, επεξήγηση υπαρχόντων προγραμμάτων, δημιουργία δοκιμών και υποστήριξη στον εντοπισμό και στη διόρθωση σφαλμάτων (Chen et al., 2021).

Ένα από τα σημαντικότερα οφέλη που προκύπτουν αφορά την αυτοματοποίηση επαναλαμβανόμενων εργασιών. Εργασίες που απαιτούσαν σημαντικό χρόνο από τον προγραμματιστή μπορούν πλέον να υποστηριχθούν ή να πραγματοποιηθούν εν μέρει από εργαλεία Τεχνητής Νοημοσύνης. Με τον τρόπο αυτό, ο επαγγελματίας μπορεί να αφιερώνει περισσότερο χρόνο σε δραστηριότητες που απαιτούν μεγαλύτερη κατανόηση του προβλήματος, σχεδιασμό και λήψη αποφάσεων. Ωστόσο, η πραγματική επίδραση των εργαλείων αυτών στην παραγωγικότητα εξαρτάται από το είδος της εργασίας, την εμπειρία του χρήστη και την ποιότητα των παραγόμενων προτάσεων και δεν πρέπει να θεωρείται ίδια σε όλες τις περιπτώσεις (Vaithilingam et al., 2022).

Παράλληλα, η χρήση της Τεχνητής Νοημοσύνης μπορεί να συμβάλει στην ταχύτερη παραγωγή ενός πρώτου αποτελέσματος. Ο προγραμματιστής μπορεί να χρησιμοποιήσει το AI-generated code ως αφετηρία και στη συνέχεια να το προσαρμόσει στις πραγματικές ανάγκες του έργου. Αυτή η μορφή υποστήριξης μπορεί να μειώσει τον χρόνο που απαιτείται για ορισμένες τυποποιημένες δραστηριότητες, χωρίς όμως να εξαλείφει την ανάγκη για ανθρώπινη αξιολόγηση. Η ευκολία και η ταχύτητα παραγωγής κώδικα δεν αποτελούν από μόνες τους εγγύηση για την ποιότητα του τελικού λογισμικού (Vaithilingam et al., 2022).

Ένα βασικό συμπέρασμα της εργασίας είναι επομένως ότι η επίδραση της Τεχνητής Νοημοσύνης στην ποιότητα του λογισμικού δεν είναι αποκλειστικά θετική ή αρνητική. Από τη μία πλευρά, η AI μπορεί να υποστηρίξει τον εντοπισμό σφαλμάτων, την παραγωγή δοκιμών, την τεκμηρίωση και τη βελτίωση του κώδικα. Από την άλλη πλευρά, ο παραγόμενος κώδικας μπορεί να περιλαμβάνει λογικά προβλήματα, ακατάλληλες τεχνικές επιλογές ή ευπάθειες ασφαλείας. Συνεπώς, το τελικό αποτέλεσμα εξαρτάται σε σημαντικό βαθμό από τον τρόπο με τον οποίο ο προγραμματιστής αξιολογεί και ενσωματώνει τις προτάσεις του συστήματος (Pearce et al., 2022).

Ιδιαίτερη σημασία έχει το ζήτημα της ασφάλειας. Το γεγονός ότι ένα σύστημα AI μπορεί να δημιουργήσει κώδικα που φαίνεται λειτουργικός δεν σημαίνει ότι αυτός είναι απαραίτητα ασφαλής. Η σχετική έρευνα έχει δείξει ότι οι προτάσεις εργαλείων παραγωγής κώδικα μπορούν σε ορισμένες περιπτώσεις να περιλαμβάνουν ευπάθειες, γεγονός που καθιστά απαραίτητο τον ανθρώπινο έλεγχο και τη χρήση κατάλληλων διαδικασιών δοκιμών και αξιολόγησης ασφαλείας (Pearce et al., 2022). Η ταχύτητα ανάπτυξης δεν πρέπει επομένως να επιτυγχάνεται εις βάρος της αξιοπιστίας και της προστασίας των δεδομένων.

Παράλληλα, η αυξανόμενη χρήση των μεγάλων γλωσσικών μοντέλων δημιουργεί ζητήματα που ξεπερνούν το καθαρά τεχνικό επίπεδο. Η προστασία εμπιστευτικού κώδικα και δεδομένων, η διαφάνεια σχετικά με τον τρόπο χρήσης των συστημάτων, η ευθύνη για τα παραγόμενα αποτελέσματα και οι περιορισμοί που προκύπτουν από τα δεδομένα εκπαίδευσης αποτελούν σημαντικά ζητήματα για την υπεύθυνη ενσωμάτωση της AI στη μηχανική λογισμικού. Η ανάπτυξη μεγάλων γλωσσικών μοντέλων έχει ήδη δημιουργήσει ευρύτερους προβληματισμούς σχετικά με την ποιότητα των δεδομένων, τις προκαταλήψεις, τη διαφάνεια και τις κοινωνικές επιπτώσεις της τεχνολογίας (Bender et al., 2021).

Ιδιαίτερο ζήτημα αποτελεί και η πνευματική ιδιοκτησία. Τα μοντέλα παραγωγής κώδικα εκπαιδεύονται σε μεγάλες συλλογές δεδομένων, οι οποίες μπορεί να περιλαμβάνουν υλικό που προστατεύεται από δικαιώματα πνευματικής ιδιοκτησίας ή διατίθεται με συγκεκριμένες άδειες χρήσης. Αυτό δημιουργεί ερωτήματα τόσο για τη χρήση του υλικού κατά την εκπαίδευση των μοντέλων όσο και για την περίπτωση κατά την οποία το παραγόμενο αποτέλεσμα παρουσιάζει ομοιότητες με υπάρχοντα έργα. Η εξέλιξη της τεχνολογίας καθιστά συνεπώς αναγκαία την ύπαρξη σαφέστερων πρακτικών και κανόνων σχετικά με τη χρήση AI-generated code μέσα σε πραγματικά έργα λογισμικού (Bender et al., 2021).

Η εργασία ανέδειξε επίσης ότι η ανάπτυξη και η λειτουργία προηγμένων μοντέλων Τεχνητής Νοημοσύνης συνδέονται με σημαντική χρήση υπολογιστικών και ενεργειακών πόρων. Η περιβαλλοντική διάσταση της AI δεν πρέπει να αγνοείται, ιδιαίτερα καθώς τα μοντέλα ενσωματώνονται σε υπηρεσίες που χρησιμοποιούνται καθημερινά από μεγάλο αριθμό χρηστών. Η προσέγγιση της Green AI αναδεικνύει την ανάγκη να αξιολογείται όχι μόνο η απόδοση ενός μοντέλου, αλλά και η υπολογιστική αποδοτικότητα και οι πόροι που απαιτούνται για την ανάπτυξη και τη χρήση της (Schwartz et al., 2020).

Μία από τις σημαντικότερες εξελίξεις για το μέλλον της μηχανικής λογισμικού είναι η μετάβαση από τα απλά εργαλεία παραγωγής κώδικα προς τους AI agents. Οι agents μπορούν να συνδυάζουν ένα γλωσσικό μοντέλο με εξωτερικά εργαλεία και να πραγματοποιούν μία ακολουθία ενεργειών για την αντιμετώπιση ενός προβλήματος. Έτσι, η AI αρχίζει να μετακινείται από την απλή παροχή προτάσεων προς την ενεργότερη συμμετοχή στη διαδικασία ανάπτυξης, όπως στην περιήγηση σε αποθετήρια, στην επεξεργασία αρχείων και στην εκτέλεση δοκιμών (Yang et al., 2024).

Παρά τη σημαντική αυτή πρόοδο, η πλήρως αυτόνομη ανάπτυξη σύνθετου λογισμικού δεν μπορεί ακόμη να θεωρηθεί δεδομένη. Πραγματικές εργασίες μηχανικής λογισμικού απαιτούν κατανόηση πολλών αρχείων, σχέσεων μεταξύ διαφορετικών τμημάτων του έργου και απαιτήσεων που δεν αποτυπώνονται πάντοτε πλήρως στον κώδικα. Η αξιολόγηση μοντέλων σε πραγματικά ζητήματα λογισμικού, όπως αυτά που περιλαμβάνονται στο SWE-bench, έχει αναδείξει τη σημαντική δυσκολία τέτοιων εργασιών σε σύγκριση με απλούστερες δοκιμές παραγωγής κώδικα (Jimenez et al., 2024).

Με βάση τα παραπάνω, η ανάπτυξη της Τεχνητής Νοημοσύνης είναι πιθανότερο να οδηγήσει σε μετασχηματισμό του επαγγέλματος του προγραμματιστή παρά σε απλή και πλήρη αντικατάστασή του. Ορισμένα καθήκοντα, κυρίως όσα είναι επαναλαμβανόμενα και μπορούν να περιγραφούν με σαφείς κανόνες, είναι δυνατό να αυτοματοποιηθούν σε μεγαλύτερο βαθμό. Παράλληλα, όμως, αυξάνεται η σημασία δραστηριοτήτων όπως η ανάλυση των απαιτήσεων, ο αρχιτεκτονικός σχεδιασμός, η αξιολόγηση της ποιότητας, η ασφάλεια και η τελική λήψη αποφάσεων (Liu et al., 2024).

Ο προγραμματιστής του μέλλοντος θα χρειάζεται επομένως να συνδυάζει τις παραδοσιακές τεχνικές γνώσεις με νέες δεξιότητες. Η γνώση προγραμματισμού εξακολουθεί να είναι σημαντική, ακριβώς επειδή ο χρήστης πρέπει να είναι σε θέση να κατανοεί και να αξιολογεί τον κώδικα που δημιουργείται από ένα σύστημα. Παράλληλα, μεγαλύτερη σημασία αποκτούν η κριτική σκέψη, η σωστή διατύπωση του προβλήματος, η κατανόηση των περιορισμών των μοντέλων και η δυνατότητα ελέγχου της ποιότητας και της ασφάλειας του παραγόμενου αποτελέσματος (Vaithilingam et al., 2022).

Η εξέλιξη αυτή οδηγεί σε ένα μοντέλο συνεργασίας ανθρώπου και Τεχνητής Νοημοσύνης. Η AI μπορεί να προσφέρει ταχύτητα, αυτοματοποίηση και δυνατότητα επεξεργασίας μεγάλου όγκου πληροφοριών, ενώ ο άνθρωπος διατηρεί την κατανόηση του ευρύτερου πλαισίου, την κριτική αξιολόγηση και την ευθύνη για τις τελικές αποφάσεις. Η σχέση αυτή μπορεί να αποδειχθεί περισσότερο αποτελεσματική από μία προσπάθεια πλήρους αυτοματοποίησης, ιδιαίτερα σε σύνθετα έργα όπου οι απαιτήσεις και οι συνέπειες των τεχνικών αποφάσεων δεν μπορούν πάντοτε να αποτυπωθούν πλήρως σε μία εντολή προς ένα μοντέλο (Liu et al., 2024).

Η υπεύθυνη αξιοποίηση της AI απαιτεί επομένως να διατηρηθεί ο άνθρωπος μέσα στη διαδικασία, ιδιαίτερα όταν πρόκειται για κρίσιμες αποφάσεις. Οι ομάδες ανάπτυξης χρειάζεται να εφαρμόζουν σαφείς διαδικασίες για τον έλεγχο του AI-generated code, την προστασία των δεδομένων και τον καθορισμό των ενεργειών που μπορούν να πραγματοποιούν περισσότερο αυτόνομα συστήματα. Η αύξηση των δυνατοτήτων της Τεχνητής Νοημοσύνης πρέπει να συνοδεύεται από αντίστοιχη ανάπτυξη των μηχανισμών εποπτείας και αξιολόγησής της (Liu et al., 2024).

Η Τεχνητή Νοημοσύνη έχει ήδη αρχίσει να αλλάζει ουσιαστικά τον τρόπο δημιουργίας λογισμικού και είναι πιθανό η επιρροή της να ενισχυθεί σημαντικά τα επόμενα χρόνια. Η πραγματική της αξία δεν βρίσκεται μόνο στην ικανότητα να παράγει περισσότερο κώδικα σε λιγότερο χρόνο, αλλά στη δυνατότητα να υποστηρίζει τον άνθρωπο σε ένα ευρύτερο σύνολο δραστηριοτήτων της μηχανικής λογισμικού. Ταυτόχρονα, οι τεχνικοί, επαγγελματικοί, νομικοί και περιβαλλοντικοί περιορισμοί δείχνουν ότι η εξέλιξη αυτή χρειάζεται να πραγματοποιηθεί με έλεγχο και υπευθυνότητα. Το πιθανότερο μέλλον δεν είναι επομένως μία διαδικασία δημιουργίας λογισμικού χωρίς προγραμματιστές, αλλά ένα περιβάλλον στο οποίο ο προγραμματιστής θα εργάζεται μαζί με την Τεχνητή Νοημοσύνη, αναθέτοντάς της περισσότερες επιμέρους εργασίες και διατηρώντας ο ίδιος τον κεντρικό ρόλο στην καθοδήγηση, στην αξιολόγηση και στην τελική ευθύνη για το λογισμικό που δημιουργείται (Liu et al., 2024).

Βιβλιογραφία

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001). *Manifesto for Agile Software Development*.

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J. & Santos, E. A. (2023). Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education (SIGCSE 2023)*, 500-506.

Becker, J., Rush, N., Barnes, N. & Rein, D. (2025). *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity*. METR.

Bender, E. M., Gebru, T., McMillan-Major, A. & Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 610-623.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York: Springer.

Bostrom, N. (2014). *Superintelligence: Paths, Dangers, Strategies*. Oxford: Oxford University Press.

Brooks, F. P. Jr. (1995). *The Mythical Man-Month: Essays on Software Engineering*. Anniversary ed. Addison-Wesley.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. & Wilkes, J. (2016). Borg, Omega, and Kubernetes. *Communications of the ACM*, 59(5), 50-57.

Chacon, S. & Straub, B. (2014). *Pro Git*. 2nd ed. Apress.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I. & Zaremba, W. (2021). *Evaluating Large Language Models Trained on Code*. arXiv:2107.03374.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-Based Software Architectures*. Doctoral dissertation, University of California, Irvine.

Gartner (2023). *Market Guide for AI-Augmented Software Development Tools*. Gartner.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press.

Hastie, T., Tibshirani, R. & Friedman, J. (2008). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer.

Humble, J. & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley.

IBM (2023). *What Is AIOps?* IBM.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O. & Narasimhan, K. (2024). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *International Conference on Learning Representations (ICLR 2024)*.

Jin, H., Huang, L., Cai, H., Yan, J., Li, B. & Chen, H. (2024). *From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future*. arXiv.

Kim, G., Humble, J., Debois, P. & Willis, J. (2021). *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. 2nd ed. IT Revolution Press.

Li, P., Yang, J., Islam, M. A. & Ren, S. (2023). *Making AI Less "Thirsty": Uncovering and Addressing the Secret Water Footprint of AI Models*. arXiv:2304.03271.

Liu, J., Wang, K., Chen, Y., Peng, X., Chen, Z., Zhang, L. & Lou, Y. (2024). *Large Language Model-Based Agents for Software Engineering: A Survey*. arXiv:2409.02977.

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.

McCarthy, J., Minsky, M. L., Rochester, N. & Shannon, C. E. (1955). *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*.

McGraw, G. (2006). *Software Security: Building Security In*. Addison-Wesley Professional.

Mell, P. & Grance, T. (2011). *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145.

METR (2026). *Measuring AI Ability to Complete Long Tasks*. Model Evaluation & Threat Research.

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S. & Xiong, C. (2022). *CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis*. arXiv:2203.13474.

Nilsson, N. J. (2009). *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press.

OpenAI (2023). *GPT-4 Technical Report*. arXiv:2303.08774.

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B. & Karri, R. (2022). Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. *2022 IEEE Symposium on Security and Privacy*, 754-768.

Peng, S., Kalliamvakou, E., Cihon, P. & Demirer, M. (2023). *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot*. arXiv:2302.06590.

Pressman, R. S. & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach*. 9th ed. McGraw-Hill Education.

Rosenblatt, F. (1957). *The Perceptron: A Perceiving and Recognizing Automaton*. Cornell Aeronautical Laboratory.

Russell, S. J. & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. 3rd ed. Pearson.

Russell, S. J. & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. 4th ed. Pearson.

Schwartz, R., Dodge, J., Smith, N. A. & Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12), 54-63.

Sommerville, I. (2011). *Software Engineering*. 9th ed. Addison-Wesley.

Strubell, E., Ganesh, A. & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3645-3650.

Tambon, F., Moradi-Dakhel, A., Nikanjam, A., Khomh, F., Desmarais, M. C. & Antoniol, G. (2025). Bugs in large language models generated code: An empirical study. *Empirical Software Engineering, 30*, 65. DOI: 10.1007/s10664-025-10614-4.

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, 59(236), 433-460.

Vaithilingam, P., Zhang, T. & Glassman, E. L. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *CHI Conference on Human Factors in Computing Systems Extended Abstracts*.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K. & Press, O. (2024). SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *Advances in Neural Information Processing Systems*.

Zhang, B., Liang, P., Zhou, X., Ahmad, A. & Waseem, M. (2023). *Practices and Challenges of Using GitHub Copilot: An Empirical Study*. arXiv.