ChatGPT

Evoluzione dell’assistenza alla programmazione e stime di produttività 1950–2026

Sintesi esecutiva

Questa sezione ricostruisce l’evoluzione dei sistemi di “programming assistance” dal 1950 al 12 febbraio 2026 e propone una **serie storica di moltiplicatori di produttività** (output utile per unità di tempo) normalizzati a un “programmatore medio anni ’50” (= 1,0×). La metrica principale è **funzionalità consegnata e verificata per mese-persona**, preferendo **Function Points** (FP) perché le metriche LOC distorcono i confronti tra linguaggi e ignorano costi “non-codice” (requisiti, design, documentazione). [1]

Risultato sintetico (modello cumulativo con incertezza esplicita): **2026 ≈ 41,6×** (P05–P95: **21,3×–81,0×**). I salti maggiori sono attribuiti a: (i) **linguaggi ad alto livello e compilatori**, (ii) **interattività + editor**, (iii) **ecosistemi (build, CI, package)**, (iv) **assistenti LLM e agenti**. [2]

Due stress-test (sensibilità): se si applica “integralmente” lo speed-up sperimentale di Copilot al ciclo end-to-end, la mediana sale a ~**53,9×**; se il salto dei linguaggi ad alto livello è più conservativo, scende a ~**27,7×**. [3]

Metriche, conversioni e metodo di stima

Definizione operativa di produttività (P): per una funzionalità fissata, P è il **rapporto di velocità**:

$$P = \frac{T_{\text{baseline}}}{T_{\text{toolchain}}}$$

dove T include coding, build/run, debug, test e verifiche minime per “consegnabile”. La scelta di FP come “unità di output” segue la motivazione economica: LOC penalizza i linguaggi ad alto livello e non misura il lavoro non-codice; FP invece consente confronti più coerenti. [1]

Perché la riduzione del tempo di scrittura non basta: nella pratica, una quota molto ampia dei costi di sviluppo è legata a **difetti (identificazione + correzione)** e a lavoro non-codice. Il NIST [4] riporta, come ordine di grandezza, che identificare e correggere difetti rappresenta ~**80%** dei costi di sviluppo. [5] Questo introduce rendimenti decrescenti: migliorare solo “typing speed” non moltiplica automaticamente l’output finale.

Stime quando mancano misure dirette: per ciascun milestone, definiamo un *incremento* plausibile (mediana + intervallo P05–P95). Gli intervalli vengono propagati con una simulazione Monte Carlo assumendo distribuzioni lognormali (indipendenza come approssimazione). I milestone con misure sperimentali (es. Copilot, autocomplete) ancorano alcuni incrementi a numeri osservati. [6]

Milestone e moltiplicatori cumulativi

Tabella: **produttività cumulativa** rispetto al baseline 1950 (=1,0×). L'intervallo è P05–P95 (incertezza aggregata). Le fonti citate supportano **(a)** data/introduzione e **(b)**, quando disponibile, evidenza quantitativa o contesto che giustifica la direzione dell'impatto.

Anno	Milestone (famiglia di strumenti)	Produttività stimata (mediana, P05–P95)	Fonti chiave
1950	Baseline: batch + machine/ assembly, tooling minimo	1,00× (1,00–1,00)	Discussione su costi fissi, assembly e limiti LOC. [1]
1957	Linguaggi ad alto livello + compilatori (era FORTRAN)	3,00× (1,89–4,75)	FORTRAN automatic coding; rilascio 1957. [7]
1961	Time-sharing & feedback interattivo (CTSS)	4,05× (2,46–6,67)	CTSS dimostrato 1961 (doc storico). [8]
1965	Editor online (QED/line editing → base per editor moderni)	4,65× (2,80–7,75)	QED come predecessore storico degli editor moderni. [9]
1968	Editing/collaborazione interattiva (NLS)	5,12× (3,04–8,63)	"Mother of all demos" e NLS. [10]
1974	Cut/Copy/Paste (editing modeless)	5,47× (3,23–9,27)	Origini cut/copy/paste in PARC (Tesler). [11]
1975	Version control (SCCS)	6,01× (3,55–10,15)	SCCS (paper storico). [12]
1978	Build automation (make)	6,74× (3,96–11,54)	Paper "make" (Feldman). [13]
1978	Static analysis classica (lint)	7,07× (4,12–12,22)	"Lint" program checker. [14]
1983	IDE su PC + compilazione rapida (Turbo Pascal)	8,47× (4,91–14,88)	Turbo Pascal come IDE+compiler integrato (data 1983). [15]
1986	Debugger simbolici mainstream (famiglia gdb/dbx)	9,22× (5,28–16,38)	Manualistica gdb e storia GNU. [16]
1995	Package repository per linguaggi (CPAN)	10,57× (6,03–18,98)	CPAN online dal 1995. [17]
1997	IDE suite GUI mainstream (Visual Studio 97)	12,16× (6,86–21,57)	Annuncio Visual Studio 97. [18]
1998	Code completion/parameter info (IntelliSense, VS6 era)	13,12× (7,38–23,37)	IntelliSense introdotta con VC6; VS6 1998. [19]
2001	IDE piattaforma estendibile (Eclipse)	14,44× (8,08–25,84)	Origini Eclipse (annuncio 2001). [20]
2001	Continuous integration server (CruiseControl)	15,88× (8,84–28,59)	CruiseControl come CI nel 2001. [21]

Anno	Milestone (famiglia di strumenti)	Produttività stimata (mediana, P05–P95)	Fonti chiave
2003	Package index centralizzato (PyPI)	16,68× (9,26–30,10)	PyPI lanciato nel 2003 (PSF). [22]
2004	Build lifecycles + dependency mgmt (Apache Maven)	18,01× (9,95–32,70)	Maven (progetto ASF; storia/ adozione). [23]
2004	Refactoring/navigation extensions (ReSharper/ Coderush)	19,45× (10,70–35,52)	Rilascio ReSharper 1.0 (JetBrains). [24]
2005	Distributed version control (Git)	21,79× (11,90–40,04)	Annuncio iniziale Git (Torvalds, 2005). [25]
2008	Installer standard (pip)	23,10× (12,57–42,53)	Post di lancio/rename a pip (2008). [26]
2010	Package manager ecosistema JS (npm)	24,27× (13,15–44,83)	Nascita npm (timeline JetBrains). [27]
2016	Standardizzazione “language intelligence” (LSP)	25,48× (13,77–47,23)	Annuncio LSP (VS Code). [28]
2021	LLM code completion preview (technical preview)	26,74× (14,42–49,66)	Lancio preview (2021). [29]
2022	LLM assistant su larga scala (GA + evidenza sperimentale)	32,09× (17,15–60,15)	GA 2022; sperimentazione: +55,8% speed (CI 21–89). [30]
2025	Agenti in terminale/IDE (Claude Code GA)	35,33× (18,64–67,06)	Claude Code “generally available” (2025). [31]
2025	Agente cloud di software engineering (Codex agent preview)	39,57× (20,45–76,40)	“Introducing Codex” (2025). [32]
2026	Multi-agent multi-modello in VCS/IDE (Agent HQ)	41,58× (21,28–81,01)	Agenti Claude+Codex in preview su GitHub (2026). [33]

Nota quantitativa importante: per l’autocomplete “classico” usiamo un ancoraggio sperimentale recente: in un esperimento controllato, l’autocomplete rende i partecipanti **~8,2% più veloci** nella completion dei task (pur senza aumentare in modo significativo il numero totale di task completati), suggerendo impatti reali ma non “esplosivi”. [34]

```
timeline
    title Evoluzione dell’assistenza alla programmazione (1950–2026)
    1950 : Batch + assembly, tooling minimo
    1957 : Linguaggi HL + compilatori (FORTRAN)
    1961 : Time-sharing (CTSS) → ciclo feedback ridotto
```

```
1965 : Editor online (QED/TECO)
1968 : NLS (demo: interazione, editing, collaborazione)
1974 : Cut/Copy/Paste
1975 : Version control (SCCS) → collaborazione più sicura
1978 : make → build automation
1978 : lint → static analysis
1983 : IDE “all-in-one” su PC (Turbo Pascal)
1997 : IDE suite (Visual Studio)
1998 : IntelliSense → code completion
2001 : Eclipse + CI (CruiseControl)
2004 : Maven + refactoring extensions
2005 : Git (DVCS)
2008 : pip + package install standard
2010 : npm + ecosistemi dipendenze
2016 : LSP → language intelligence standardizzata
2021 : LLM code completion (preview)
2022 : LLM assistant (adozione ampia + evidenza sperimentale)
2025 : Agenti (Claude Code, Codex agent)
2026 : Multi-agent multi-modello in VCS/IDE (Agent HQ)
```

Serie temporale, grafico e modelli di crescita

Trend della produttività stimata con bande di incertezza

Il grafico usa scala logaritmica: in presenza di crescita approssimativamente esponenziale, i punti tendono ad allinearsi. Nel nostro dataset, un fit esponenziale su $\log(P)$ fornisce **$R^2 \approx 0{,}956$**; un modello alternativo con **curvatura** (quadratico su $\log(P)$, proxy di rallentamento) migliora leggermente (**$R^2 \approx 0{,}965$**, AIC migliore). Questo è coerente con l’idea di **costi fissi e lavoro non-codice** che attenuano i benefici marginali dei tool (non tutto il tempo è “accelerabile”). [35]

Dataset esportabile

```
sandbox:/mnt/data/
ai_programming_assistance_productivity_multipliers_1950_2026.csv
sandbox:/mnt/data/ai_programming_assistance_productivity_trend_1950_2026.png
```

Gap, limiti e raccomandazioni di raccolta dati

Le stime tra 1950–2000 sono inevitabilmente “sparse”: molte introduzioni (editor, build, VCS) hanno impatti reali ma raramente misurati in RCT moderni. La parte 2021–2026 è più misurabile (es. studio controllato di Copilot), ma resta il problema della **generalizzabilità**: in un trial, Copilot riduce il tempo di completamento del task del 55,8% (CI 21–89), ma l’effetto end-to-end dipende da code review, debug, qualità e policy aziendali. [36]

Per rafforzare la base evidenziale (utile per un talk introduttivo ma “analitico”), le azioni più ad alto rendimento sarebbero: - **Benchmark longitudinali FP/PM** su classi di progetto comparabili (stessa “funzionalità” e qualità) per stimare trend reali evitando LOC. [1]
- **RCT industriali** su attività end-to-end (feature+test+review+merge), con logging dei tempi e misure di qualità (bug, revert, incidenti), analoghi al disegno sperimentale già usato per Copilot. [37]
- **Studi specifici sugli agenti** (terminal/IDE/VCS), perché il salto 2025–2026 riguarda soprattutto orchestrazione (task parallel, PR automation) più che autocomplete. [38]

---

[1] [35] https://www.ifpug.org/wp-content/uploads/2017/04/IYSM.-Thirty-years-of-IFPUG.-Software-Economics-and-Function-Point-Metrics-Capers-Jones.pdf
https://www.ifpug.org/wp-content/uploads/2017/04/IYSM.-Thirty-years-of-IFPUG.-Software-Economics-and-Function-Point-Metrics-Capers-Jones.pdf

[2] [7] https://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/704/FORTRAN_paper_1957.pdf
https://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/704/FORTRAN_paper_1957.pdf

[3] [6] [36] [37] https://ar5iv.org/pdf/2302.06590
https://ar5iv.org/pdf/2302.06590

[4] [26] https://ianbicking.org/blog/2008/10/pyinstall-is-dead-long-live-pip.html
https://ianbicking.org/blog/2008/10/pyinstall-is-dead-long-live-pip.html

[5] https://www.nist.gov/document/samate-document-greg-tasseys-summary-pdf-nists-2002-report-economic-impacts-inadequate
https://www.nist.gov/document/samate-document-greg-tasseys-summary-pdf-nists-2002-report-economic-impacts-inadequate

[8] https://people.csail.mit.edu/saltzer/Multics/CTSS-Documents/CTSS_50th_anniversary_web_03.pdf
https://people.csail.mit.edu/saltzer/Multics/CTSS-Documents/CTSS_50th_anniversary_web_03.pdf

[9] https://www.nokia.com/bell-labs/about/dennis-m-ritchie/qed.html
https://www.nokia.com/bell-labs/about/dennis-m-ritchie/qed.html

[10] https://www.facebook.com/groups/779220482206901/posts/24123768100658810/
https://www.facebook.com/groups/779220482206901/posts/24123768100658810/

[11] https://en.wikipedia.org/wiki/Bram_Moolenaar
https://en.wikipedia.org/wiki/Bram_Moolenaar

[12] https://www.computer.org/csdl/journal/ts/1975/04/06312866/13rRUynZ5ps
https://www.computer.org/csdl/journal/ts/1975/04/06312866/13rRUynZ5ps

[13] https://cgi.csc.liv.ac.uk/~coopes/comp319/2016/papers/feldman79make.pdf
https://cgi.csc.liv.ac.uk/~coopes/comp319/2016/papers/feldman79make.pdf

[14] https://wolfram.schneider.org/bsd/7thEdManVol2/lint/lint.pdf
https://wolfram.schneider.org/bsd/7thEdManVol2/lint/lint.pdf

[15] https://en.wikipedia.org/wiki/Turbo_Pascal
https://en.wikipedia.org/wiki/Turbo_Pascal

[16] https://www.sourceware.org/gdb/current/onlinedocs/gdb.pdf
https://www.sourceware.org/gdb/current/onlinedocs/gdb.pdf

17 https://www.cpan.org/index.html
https://www.cpan.org/index.html

18 https://news.microsoft.com/source/1997/01/28/microsoft-announces-visual-studio-97-a-comprehensive-suite-of-microsoft-visual-development-tools/
https://news.microsoft.com/source/1997/01/28/microsoft-announces-visual-studio-97-a-comprehensive-suite-of-microsoft-visual-development-tools/

19 https://devblogs.microsoft.com/cppblog/intellisense-history-part-1/
https://devblogs.microsoft.com/cppblog/intellisense-history-part-1/

20 https://wiki.eclipse.org/FAQ_Where_did_Eclipse_come_from%3F
https://wiki.eclipse.org/FAQ_Where_did_Eclipse_come_from%3F

21 https://martinfowler.com/articles/go-interview.html
https://martinfowler.com/articles/go-interview.html

22 https://blog.pypi.org/posts/2024-03-20-announcing-a-pypi-support-specialist/
https://blog.pypi.org/posts/2024-03-20-announcing-a-pypi-support-specialist/

23 https://maven.apache.org/docs/history.html
https://maven.apache.org/docs/history.html

24 https://blog.jetbrains.com/blog/2004/07/21/pr_210704/
https://blog.jetbrains.com/blog/2004/07/21/pr_210704/

25 https://yarchive.net/comp/linux/git.html
https://yarchive.net/comp/linux/git.html

27 https://www.jetbrains.com/lp/javascript-25/
https://www.jetbrains.com/lp/javascript-25/

28 https://code.visualstudio.com/blogs/2016/06/27/common-language-protocol
https://code.visualstudio.com/blogs/2016/06/27/common-language-protocol

29 https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/
https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

30 https://github.blog/news-insights/product-news/github-copilot-is-generally-available-to-all-developers/
https://github.blog/news-insights/product-news/github-copilot-is-generally-available-to-all-developers/

31 38 https://www.anthropic.com/news/claude-4
https://www.anthropic.com/news/claude-4

32 https://openai.com/index/introducing-codex/
https://openai.com/index/introducing-codex/

33 https://github.blog/changelog/2026-02-04-claude-and-codex-are-now-available-in-public-preview-on-github/
https://github.blog/changelog/2026-02-04-claude-and-codex-are-now-available-in-public-preview-on-github/

34 https://cseweb.ucsd.edu/~mcoblenz/assets/pdf/fse24-autocomplete.pdf
https://cseweb.ucsd.edu/~mcoblenz/assets/pdf/fse24-autocomplete.pdf