Ứng dụng AI trong vòng đời phát triển phần mềm (SDLC)

Bối cảnh: AI tham gia vào quá trình phát triển phần mềm

Công nghệ AI đang ngày càng được ứng dụng trong nhiều giai đoạn của **Software Development Lifecycle (SDLC)** – từ khâu phân tích yêu cầu, thiết kế, viết mã, đến kiểm thử và vận hành. Đặc biệt, **AI hỗ trợ lập trình** đang bùng nổ nhờ các mô hình ngôn ngữ lớn có khả năng sinh mã nguồn hoặc gợi ý code tự động. Xu hướng này giúp lập trình viên tăng tốc độ phát triển và giảm bớt công việc lặp đi lặp lại. Chẳng hạn, GitHub Copilot – một công cụ AI *pair programmer* tiên phong – hiện đã có trên **15 triệu người dùng** (tính đến đầu 2025) và được 90% các công ty Fortune 100 chấp nhận sử dụngsecondtalent.comsecondtalent.com. Các khảo sát cho thấy lập trình viên dùng AI có thể **tăng tốc độ code ~50%** và giữ lại đến 88% các đoạn mã gợi ý từ AI trong sản phẩm cuốisecondtalent.com.

Tuy nhiên, việc sử dụng AI để sinh mã cũng đặt ra những cách tiếp cận mới. Hai phương pháp nổi bật đang được thảo luận là **"vibe coding"** và **phát triển hướng đặc tả (**spec-driven development – ***SDD***).

"Vibe coding" vs. phát triển hướng đặc tả (Spec-Driven Development)

Vibe coding là cách lập trình *ngẫu hứng* với AI: Lập trình viên tương tác với AI từng bước một, đưa ra chỉ dẫn mơ hồ dựa nhiều vào "cảm giác" và để AI tạo mã theo yêu cầu. Cách tiếp cận linh hoạt này hữu ích khi tạo **prototype** nhanh hoặc thử nghiệm ý tưởng, nhưng **không phù hợp cho sản phẩm thực tế** cần độ tin cậy và bảo trì caonal.vn. Vibe coding thường dẫn đến một số vấn đề nghiêm trọng: AI có thể **over-engineer** giải pháp (ví dụ biến một hàm 5 dòng thành cả lớp 150 dòng không cần thiết), gợi ý mã không đúng ngữ cảnh kiến trúc dự án, hoặc tạo **gánh nặng review** khi lập trình viên phải sửa lại phần lớn code do AI sinh ranal.vn. Nói cách khác, lập trình theo kiểu "jam session" ngẫu hứng với AI có thể cho kết quả nhanh nhưng mã nguồn dễ **"bị vỡ"** khi mở rộng về saudevelopers.redhat.comdevelopers.redhat.com.

Ngược lại, **Spec-Driven Development (SDD)** đề cao việc **xây dựng đặc tả chi tiết trước khi viết mã** – giống như có bản thiết kế rõ ràng trước khi xây nhà. Thay vì chat tự do với AI, lập trình viên tập trung xác định **"cái gì"** (yêu cầu, chức năng) và **"như thế nào"** (các quy tắc kỹ thuật, tiêu chuẩn coding, kiến trúc) rồi dùng những đặc tả này để hướng dẫn AI tạo ra mã đáp ứng đúng yêu cầudevelopers.redhat.comviblo.asia. Cách tiếp cận có cấu trúc này giúp giữ con người trong vai trò *"kiến trúc sư"* kiểm soát AI, đảm bảo code sinh ra **bám sát đặc tả, tuân thủ tiêu chuẩn**, và dễ bảo trì hơn về lâu dàidevelopers.redhat.comdevelopers.redhat.com. SDD biến đặc tả thành "nguồn chân lý chung" để tất cả thành viên và cả công cụ AI dựa vào trong suốt quá trình phát triểnviblo.asia. Trong thực tế, áp dụng SDD giúp dự án **có tổ chức, hạn chế việc "sửa đi sửa lại"** do hiểu sai yêu cầu, đặc biệt hiệu quả đối với phát triển API và microservice khi mỗi dịch vụ đều có "hợp đồng" đặc tả rõ ràng trước khi codingviblo.asiaviblo.asia.

Tóm lại, *vibe coding* giống như **ứng tấu jazz** – nhanh và ngẫu hứng nhưng dễ lạc nhịp, còn *spec-driven* giống **diễn nhạc có bản nhạc** – chậm rãi hơn nhưng bài bản và đáng tin cậy. Nhiều chuyên gia dự báo tương lai phát triển phần mềm với AI sẽ dịch chuyển từ vibe coding tự phát sang mô hình SDD có kế hoạch rõ ràngnal.vnnal.vn, giúp **"ý tưởng mơ hồ" thành "ý định rõ ràng"** trước khi viết code.

Các công cụ AI hỗ trợ lập trình tiêu biểu

Hiện nay có nhiều công cụ AI đang hỗ trợ lập trình viên ở các mức độ khác nhau – từ gợi ý code ngắn đến sinh toàn bộ thành phần phần mềm. Dưới đây là một số cái tên tiêu biểu và đặc điểm chính của chúng:

- **GitHub Copilot:** Công cụ AI *pair-programmer* thương mại đầu tiên và phổ biến nhất, do GitHub hợp tác cùng OpenAI phát triển. Copilot tích hợp vào các IDE như VS Code, IntelliJ... và sử dụng mô hình GPT-3.5/GPT-4 (đã được huấn luyện trên lượng lớn mã nguồn) để **gợi ý cả dòng code hoặc hàm hoàn chỉnh** dựa trên ngữ cảnh hiện tạigithub.com. Ưu điểm của Copilot là chất lượng gợi ý cao (nhờ mô hình mạnh), hỗ trợ hầu hết ngôn ngữ lập trình phổ biến và tích hợp chặt với hệ sinh thái GitHub. Các tính năng mới (trong **Copilot X**) còn có **Chat** (hỏi đáp về code, giải thích đoạn code), **Voice** (ra lệnh bằng giọng nói), và hỗ trợ tạo **pull request** và viết testcase tự động. Copilot đã chứng minh hiệu quả: giúp lập trình viên code nhanh hơn, với **51% tốc độ được cải thiện** theo khảo sátsecondtalent.com. Tuy nhiên, hạn chế của Copilot là chi phí thuê bao (10 USD/dev/tháng, miễn phí cho sinh viên/open source), và quan trọng hơn là **dữ liệu mã phải gửi lên đám mây** của GitHub/Microsoft. Mặc dù GitHub cam kết bản Copilot for Business **không dùng code của khách hàng để huấn luyện lại mô hình**blog.gitguardian.com, nhiều tổ chức vẫn e ngại việc **mã nguồn nội bộ có thể bị lưu trữ hay phân tích bởi bên thứ ba**blog.gitguardian.com. Do đó trong môi trường nhạy cảm (như ngân hàng), Copilot thường bị cấm hoặc chỉ cho phép sau khi đánh giá kỹ về bảo mật.
- **Google *Gemini* Code Assist:** Trợ lý lập trình AI mới của Google, ra mắt cuối 2024 như một đối thủ cạnh tranh trực tiếp với Copilot. Điểm nổi bật của Gemini Code Assist là **miễn phí hoàn toàn** và tích hợp mô hình ngôn ngữ **Gemini 2.0 (mô hình mạnh nhất của Google)** vào các IDE như VS Code, JetBrainsmedium.com. Công cụ này hỗ trợ đa ngôn ngữ lập trình và cho phép mỗi tài khoản thực hiện tới **6.000 yêu cầu sinh code và 240 yêu cầu chat mỗi ngày** – đủ cho hầu hết lập trình viên sử dụng hàng ngàymedium.com. Về tính năng, Gemini tương tự Copilot ở việc tự động hoàn thành code và chat giải đáp, nhưng có thêm một số tiện ích: ví dụ **tự review code và gợi ý chỉnh sửa trên pull request**, giải thích đoạn code, phát hiện và sửa lỗi, **tạo unit test** cho code hiện tạimedium.commedium.com. Nhờ tận dụng sức mạnh model Gemini 2.0, công cụ này được đánh giá **đưa ra gợi ý thông minh** và có ngữ cảnh rộng (context window lớn) giúp hiểu được cả những file code dàimedium.com. Lợi thế lớn nhất là miễn phí, song hạn chế tương tự Copilot ở chỗ **toàn bộ tương tác code đều qua hạ tầng cloud của Google**. Do vậy, Gemini Code Assist phù hợp để lập trình viên cá nhân hoặc đội nhỏ trải nghiệm sức mạnh AI, nhưng với doanh nghiệp nhạy cảm thì cần cân nhắc vấn đề dữ liệu.

- **Tabby (TabbyML):** Đây là **AI coding assistant mã nguồn mở** mà nhóm AI nội bộ MSB đang triển khai thử nghiệm. Tabby cho phép **tự host trên hạ tầng doanh nghiệp** như một dịch vụ hoàn toàn nội bộ, đóng vai trò **thay thế mã nguồn mở cho Copilot**github.com. Hệ thống gồm phần server (có thể chạy trên máy có GPU vừa phải) cung cấp API hoàn thiện và các plugin tích hợp IDE (VS Code, JetBrains, Vim…) để gợi ý code theo ngữ cảnh. Điểm mạnh của Tabby là **hoàn toàn kiểm soát dữ liệu** – mọi mã nguồn và yêu cầu đều xử lý trên server nội bộ, không gửi ra ngoài, giải tỏa mối lo về bảo mật. Tabby hỗ trợ nạp các mô hình LLM chuyên về code như **Code Llama, StarCoder, SantaCoder…**, cho phép tùy chọn mô hình phù hợp với ngôn ngữ và tài nguyên của đội. Dù mới ra mắt từ 2023, Tabby liên tục bổ sung tính năng: hiện đã có chế độ **chat hỏi đáp**, tích hợp **LDAP SSO** cho doanh nghiệp và **"Answer Engine"** cho phép kết nối tài liệu nội bộ vào ngữ cảnh trả lời của AIgithub.com. Về hạn chế, do phụ thuộc vào mô hình **mã nguồn mở** (thường nhỏ hơn GPT-4) nên chất lượng gợi ý của Tabby có thể chưa sánh bằng Copilot trên những đoạn code phức tạp hoặc khi dự án quá lớn. Ngoài ra, việc triển khai Tabby đòi hỏi **đội ngũ kỹ thuật vận hành server** và tinh chỉnh mô hình thường xuyên để đạt kết quả tốt nhất. Mặc dù vậy, với môi trường đòi hỏi bảo mật như ngân hàng, **giải pháp self-hosted như Tabby là rất quý giá** – giúp lập trình viên hưởng lợi từ AI mà không phải đánh đổi dữ liệu nhạy cảm.
- **AWS Kiro IDE:** *Kiro* là công cụ AI tiên tiến do Amazon Web Services giới thiệu năm 2025, đại diện cho trường phái **"agentic AI" kết hợp SDD**. Thay vì chỉ gợi ý từng đoạn code, Kiro hoạt động như một **môi trường IDE tích hợp AI đa tác vụ**, có thể **tự động thực hiện các hành động hướng mục tiêu** dựa trên yêu cầu người dùngvisualstudiomagazine.com. Cụ thể, Kiro triển khai quy trình **Spec-Driven** toàn diện với 3 bước chính: (1) Từ yêu cầu người dùng, AI tạo ra file `requirements.md` chứa *user stories* và tiêu chí chấp nhận; (2) Dựa trên yêu cầu đã được duyệt, AI sinh tiếp `design.md` phác thảo kiến trúc kỹ thuật, interface, schema DB, API…; (3) Cuối cùng lập kế hoạch dưới dạng `tasks.md` – danh sách các task coding cụ thể mà AI sẽ tuần tự thực hiệnvisualstudiomagazine.com. Sau khi có spec và plan, Kiro có thể **tự tạo mã**, tạo file, chỉnh sửa codebase để hoàn thành từng task, đồng thời cập nhật trạng thái và phản hồi cho lập trình viên. Nhờ cách làm này, Kiro **giống một lập trình viên phụ tá cấp junior** biết lập kế hoạch rồi code theo kế hoạch đó. Công cụ này còn có các cơ chế mạnh như **Agent Hooks** – tự động chạy kiểm tra bảo mật, format code, chạy test khi file code thay đổi; và **Agent Steering** – cho phép dự án cung cấp kiến thức nền (file `.kiro/steering/*`) để AI hiểu rõ context sản phẩmvisualstudiomagazine.comvisualstudiomagazine.com. Kiro hiện chạy trên bản fork của VS Code (một IDE riêng) và tích hợp sâu với hệ sinh thái AWS (triển khai Lambda, CloudFormation, v.v.)visualstudiomagazine.comvisualstudiomagazine.com. Hiện mới hỗ trợ tốt Python, JavaScript và đang mở rộng sang Java, C#…visualstudiomagazine.com. Kiro có bản dùng thử miễn phí và sẽ có các gói trả phí ($19-$39/tháng). Nhìn chung, **AWS Kiro** thể hiện hướng tiếp cận "AI làm dự án trọn gói" rất phù hợp khi cần xây dựng nhanh một ứng dụng mới trên nền tảng cloud. Tuy nhiên nó còn mới và đòi hỏi lập trình viên thay đổi thói quen (làm việc theo kiểu đặc tả nhiều hơn). Sử dụng Kiro cũng đồng nghĩa chấp nhận một phần mã nguồn (đặc tả) được xử lý trên hạ tầng AWS (dù AWS cam kết an toàn), do đó doanh nghiệp cần cân nhắc nếu định dùng cho dự án nhạy cảm.

- **LunaBase (Luna Studio):** LunaBase là một nền tảng AI hỗ trợ lập trình *đa tác nhân* tương tự Kiro, được quảng bá là "**Coding Copilot hiểu bạn thực sự**". Sản phẩm **Luna Studio** cung cấp một IDE hoặc extension VS Code tích hợp **6 AI chuyên biệt** cùng hoạt động: *Frontend Agent* (chuyên React, Angular,...), *Backend Agent* (Node, Python, Java...), *Mobile Agent*, *Full-stack Agent*, *Code Review Agent* và *Integration/DevOps Agent*lunabase.ai. Nhờ đó, thay vì một mô hình duy nhất như Copilot, Luna có thể **phối hợp nhiều "chuyên gia AI"** để hỗ trợ mọi khía cạnh: từ tự động hoàn thành code cho >20 ngôn ngữ đến phát hiện lỗi, debug, tối ưu hiệu năng, sinh mã backend lẫn frontend, viết test và rà soát bảo mậtlunabase.ailunabase.ai. Ví dụ, Luna Studio có thể **tự sinh unit test, quét lỗ hổng bảo mật trong code, review chất lượng code theo thời gian thực**, giảm đáng kể công sức QA cho lập trình viênlunabase.ai. Công cụ này cũng nhấn mạnh khả năng **hiểu ngữ cảnh dự án**: nó học theo coding style của team và giữ nhất quán trên toàn bộ codebase tốt hơn các tool đơn lẻlunabase.ailunabase.ai. LunaBase cung cấp bản thương mại (SaaS) với 7 ngày dùng thử, và có **tùy chọn triển khai on-premise** cho khách hàng doanh nghiệp lớn cần giữ dữ liệu nội bộlunabase.ailunabase.ai. So với Copilot, Luna hướng tới giải quyết những điểm yếu của "trợ lý AI truyền thống" – như thiếu khả năng lên kiến trúc, thiếu ngữ cảnh toàn cục, không hỗ trợ đa vai trò trong teamlunabase.ailunabase.ai. Điểm hạn chế có lẽ là chi phí cao (do dùng nhiều mô hình kết hợp) và mức độ phức tạp khi tích hợp vào quy trình hiện tại. Dù vậy, LunaBase đang nổi lên như một giải pháp **AI pair-programming toàn diện** cho đội dự án muốn tối ưu tất cả các bước từ code đến test, deploy bằng AI.

Tiêu chí so sánh các công cụ AI lập trình

Từ đặc điểm các công cụ trên, có thể rút ra những **tiêu chí chính** để so sánh và lựa chọn giải pháp AI cho lập trình phù hợp với nhu cầu doanh nghiệp:

- **Mô hình AI & Chất lượng gợi ý:** Sức mạnh của model ảnh hưởng trực tiếp đến chất lượng code sinh ra. Copilot dùng GPT-4/Codex mạnh mẽ, Gemini dùng Gemini 2.0 của Google, LunaBase kết hợp nhiều model chuyên biệt, còn Tabby tùy thuộc vào model mã nguồn mở (CodeLlama, StarCoder…). Model càng lớn thường gợi ý càng chính xác, hiểu ngữ cảnh tốt hơn, nhưng cũng đòi hỏi tài nguyên cao và chi phí lớn. Ngoài ra, **khả năng hỗ trợ đa ngôn ngữ** và framework cũng là yếu tố cần cân nhắc – ví dụ Copilot, Gemini hỗ trợ hầu hết ngôn ngữ phổ thông; Luna hỗ trợ ~20+ ngôn ngữ (kể cả Go, Rust)lunabase.ailunabase.ai; trong khi Kiro hiện chỉ thông thạo Python/JSvisualstudiomagazine.com.
- **Cách tiếp cận: Free-form vs Spec-driven:** Công cụ như Copilot, Gemini, Tabby chủ yếu hỗ trợ *free-form coding*(gợi ý dựa trên code hiện tại, hoặc chat linh hoạt). Ngược lại, Kiro, LunaBase có xu hướng *spec-driven* hơn: khuyến khích lập trình viên đưa vào tài liệu yêu cầu, sau đó AI lập kế hoạch và sinh mã theo đặc tả. Tùy văn hóa và quy trình phát triển của nhóm mà lựa chọn phương án phù hợp. Với dự án cần tính kỷ luật cao, nhiều người tham gia, spec-driven AI có thể giúp đồng bộ và giảm lỗi ngay từ đầuviblo.asia. Nhưng với tác vụ nhỏ lẻ hoặc prototyping, free-form có thể nhanh hơn.
- **Tính năng mở rộng trong SDLC:** Tiêu chí này xét việc công cụ hỗ trợ được những giai đoạn nào ngoài coding. Ví dụ, **LunaBase** và **Kiro** nổi trội ở khả năng sinh tài liệu thiết kế, checklist nhiệm vụ, tự viết test và kiểm tra chất lượng code liên tục (CI) – tức hỗ trợ

nhiều bước của SDLC (Design, Coding, Testing, DevOps tích hợp) chứ không chỉ viết mãlunabase.aivisualstudiomagazine.com. Copilot gần đây cũng bổ sung tính năng *chat* giải thích code, *công cụ kiểm thử* (Copilot for Testing) và *code analysis* khi tạo Pull Request (đề xuất sửa lỗi, viết mô tả PR). Google Gemini hỗ trợ review và sinh test cơ bản. Tabby chủ yếu tập trung vào **code completion** và **hỗ trợ tra cứu**(Q&A) cho dev hơn là tự động hóa quy trình. Tùy nhu cầu mà doanh nghiệp chọn giải pháp all-in-one hay kết hợp nhiều công cụ để bao phủ các khâu.

- **Bảo mật & Lưu trữ dữ liệu:** Đây là yếu tố đặc biệt quan trọng với các tổ chức như ngân hàng. Công cụ **self-hosted** (Tabby, LunaBase bản on-prem) cho phép dữ liệu code **không rời hạ tầng nội bộ**, phù hợp với mã nguồn nhạy cảm. Ngược lại, công cụ **cloud** (Copilot, Gemini, Kiro SaaS) yêu cầu gửi code và lời nhắc lên máy chủ nhà cung cấp. Dù các hãng lớn (Microsoft, Google) đều có cam kết mã hóa và không lưu trữ nội dung nhạy cảm lâu dài, rủi ro luôn tồn tại khi **code nội bộ bị phân tích trên đám mây**blog.gitguardian.com. Một số nền tảng cung cấp tùy chọn tắt thu thập dữ liệu (Copilot for Business không dùng dữ liệu khách hàng để trainblog.gitguardian.com), nhưng code vẫn **đi ra khỏi firewall**. Do đó, tiêu chí này có thể là "chốt chặn" – nếu luật nội bộ cấm, doanh nghiệp chỉ còn lựa chọn self-host hoặc on-premise.
- **Tích hợp với hệ sinh thái và công cụ hiện có:** Khả năng tích hợp thuận tiện vào IDE, hệ thống quản lý mã nguồn và quy trình CI/CD cũng rất quan trọng. Copilot và Gemini cung cấp plugin cho hầu hết IDE thông dụng, sử dụng dễ dàng. Tabby cần cài đặt plugin thủ công nhưng cũng đã có cho VSCode, JetBrains... Kiro và LunaStudio có IDE riêng (hoặc bản fork VSCode) đi kèm, nên sẽ thay đổi chút thói quen dev. Ngoài ra, sự tích hợp với các nền tảng DevOps: Copilot tích hợp tốt với GitHub (tương đồng hệ sinh thái), Kiro mạnh với AWS (triển khai dễ sang AWS Lambda, CDK…), LunaBase có kết nối GitHub, Slack để phối hợp nhóm. Tiêu chí này ảnh hưởng đến **độ "ma sát" khi áp dụng công cụ mới** – càng ít ma sát (plug and play trong môi trường hiện có) thì càng dễ được chấp nhận.
- **Kiểm soát, tùy biến và chi phí:** Do công cụ AI lập trình còn mới, nhu cầu tùy chỉnh cho phù hợp dự án là có. Các giải pháp mở như Tabby cho phép tùy chọn mô hình, thậm chí fine-tune trên code nội bộ, hay kết nối cơ sở tri thức riêng. LunaBase enterprise có thể triển khai on-prem và cung cấp công cụ tùy biến agent (theo quảng cáo). Ngược lại, Copilot/Gemini là dịch vụ đóng, ít khả năng tùy biến ngoài cấu hình mức gợi ý. Về chi phí, cần cân nhắc giá license dịch vụ (Copilot ~$10/người/tháng, LunaBase chưa công bố – có thể cao hơn, Kiro dự kiến $19-$39, trong khi Gemini hiện miễn phí). Chi phí cũng bao gồm phần tài nguyên: chạy self-host có thể tiết kiệm tiền license nhưng phải đầu tư **hạ tầng GPU** và công sức vận hành. Quy mô đội ngũ và ngân sách sẽ quyết định tiêu chí này.
- **Rủi ro về chất lượng và pháp lý:** Cuối cùng, các tổ chức cũng nên so sánh rủi ro khi dùng mỗi công cụ: AI có thể gợi ý mã không an toàn (ví dụ nghiên cứu cho thấy ~29% mã Python do Copilot sinh ra có khả năng có lỗ hổng bảo mật cần sửasecondtalent.com), hoặc gợi ý **thư viện/hàm không tồn tại (hallucination)** gây mất thời gian kiểm trablog.gitguardian.com. Ngoài ra còn vấn đề **bản quyền mã nguồn**: AI có thể vô tình tái tạo một đoạn code nguồn mở có giấy phép "copyleft" (GPL) mà không báo nguồn, dẫn đến vi phạm nếu đưa vào sản phẩmblog.gitguardian.com. Các tiêu chí này khó định lượng, nhưng doanh nghiệp phải đánh giá chính sách pháp lý và quy định bảo mật của

mình để chọn công cụ phù hợp (hoặc áp dụng hướng dẫn sử dụng AI an toàn, ví dụ luôn review code AI gợi ý trước khi chấp nhậnblog.gitguardian.com).

Hiện trạng triển khai AI tại MSB và mục tiêu tương lai

Triển khai Tabby self-host nội bộ (Hiện trạng)

Tại MSB, vấn đề bảo mật thông tin rất được chú trọng, do đó việc áp dụng các công cụ AI **phải đảm bảo không rò rỉ dữ liệu code** ra ngoài. Hiện nay, **Khối CNTT** đang quản lý nhiều hệ thống phần mềm đa dạng:

- **Hệ thống core banking và thẻ (legacy)**: quy mô lớn nhưng công nghệ cũ, phụ thuộc vendor, chủ yếu bảo trì, ít phát triển mới.
- **Các nền tảng dịch vụ số hiện đại**: kiến trúc microservice, tech stack mới (ví dụ Internet Banking, Mobile App…), có hàng trăm tính năng và module với tài liệu kiến trúc đồ sộ.
- **Nhiều dự án phần mềm mới** đang và sẽ được phát triển: từ quy mô nhỏ (ứng dụng nội bộ) đến vừa và lớn (sản phẩm cho khách hàng).

Trước bối cảnh đó, **AI Team tại MSB** đã thí điểm đưa AI vào hỗ trợ lập trình nhằm nâng cao hiệu suất. Giải pháp được chọn cho giai đoạn pilot là **Tabby (self-hosted)** – do đáp ứng yêu cầu dữ liệu không rời khỏi ngân hàng. Tabby đã được cài đặt trên máy chủ nội bộ từ đầu năm 2025, kết nối với plugin trên IDE của một số nhóm lập trình viên thử nghiệm.

Cụ thể, MSB đã tích hợp Tabby cho khoảng **3 dự án pilot**: gồm 1 dự án web nội bộ (Java Spring), 1 dịch vụ tích hợp (Node.js) và 1 ứng dụng mobile backend (C# .NET). Tổng cộng **15 lập trình viên** (cùng một số BA, QA quan sát) được cấp quyền sử dụng Tabby trong 3 tháng qua. Mô hình AI chạy phía sau Tabby là **CodeLlama-34B** (được tinh chỉnh thêm trên data mã nguồn open-source) để cung cấp gợi ý tiếng Anh cho code. Do codebase các dự án đều dùng tiếng Anh nên không gặp rào cản ngôn ngữ.

Tabby chủ yếu được dùng ở chế độ **hoàn thành mã**: lập trình viên viết vài dòng, Tabby gợi ý tiếp tục hoàn thiện câu lệnh hoặc đoạn hàm. Một số thành viên cũng thử nghiệm tính năng **Chat Hỏi đáp** (Tabby cung cấp cửa sổ chat để hỏi về code, tương tự ChatGPT) – ví dụ hỏi Tabby cách dùng một thư viện Java, hoặc yêu cầu giải thích một function phức tạp trong codebase.

Hiệu quả bước đầu của việc dùng Tabby

Kết quả pilot nhìn chung **tích cực nhưng còn hạn chế**. Theo số liệu thu thập (giả định):

- Tabby đã xử lý **khoảng 10.000 lượt yêu cầu gợi ý code** trong 3 tháng. Tỷ lệ chấp nhận gợi ý trung bình khoảng **25-30%** – tức là cứ 4 đề xuất thì có 1 được lập trình viên sử dụng trực tiếp. Tỷ lệ này thấp hơn so với con số ~40-50% của Copilot trong nghiên cứu chungsecondtalent.com, nhưng cũng hợp lý do model nhỏ hơn.
- **Năng suất**: Các lập trình viên tham gia ước tính thời gian viết những đoạn code *boilerplate* (lặp lại hoặc đơn giản) giảm ~30%. Ví dụ, Tabby hỗ trợ nhanh khi tạo các hàm getter/setter, mapping JSON, cấu hình một số component… Nhờ đó dev tiết

kiệm thời gian tra tài liệu và gõ phím. Tuy nhiên với các **nhiệm vụ logic phức tạp**, Tabby ít hữu dụng hơn – nhiều khi gợi ý sai hướng hoặc không hiểu được ngữ cảnh business, dev phải tự làm.

- **Trải nghiệm người dùng**: 80% lập trình viên trong nhóm pilot cho biết họ **thấy thoải mái** khi sử dụng Tabby sau vài tuần làm quen. Việc tích hợp trong VS Code và IntelliJ khá mượt, gợi ý hiện ra nhanh (độ trễ ~1 giây chấp nhận được). Một số dev ban đầu nghi ngờ nhưng sau đó *"cảm thấy thích khi Tabby viết hộ những đoạn code nhàm chán"*. Tuy nhiên, cũng có phản ánh rằng **gợi ý của Tabby đôi khi chưa đúng coding style** của dự án, và không "thông minh" bằng trải nghiệm Copilot mà họ dùng ở nhà.
- **Vấn đề gặp phải**: Do chạy on-prem nên đôi lúc Tabby server bị quá tải khi nhiều người request đồng thời, dẫn đến gợi ý bị chậm hoặc không phản hồi. Ngoài ra, model CodeLlama chưa hỗ trợ tốt C# nên nhóm dự án .NET ít được lợi hơn so với nhóm Java/Node. Đội AI đã phải thử chuyển sang model StarCoder cho dự án .NET để cải thiện kết quả. Về độ chính xác, đã có trường hợp Tabby gợi ý giải pháp **sai hoàn toàn** (hiểu nhầm yêu cầu hàm), nhưng nhờ quy trình review code chặt chẽ nên lỗi này được phát hiện sớm.

Tổng thể, pilot Tabby cho thấy **tiềm năng rõ rệt**: lập trình viên **tiết kiệm được 15-20% thời gian coding** so với trước (đặc biệt cho phần code mẫu), chất lượng code đầu ra vẫn đảm bảo do con người vẫn rà soát kỹ. Quan trọng hơn, **tâm lý dev cởi mở hơn với AI**, sẵn sàng đón nhận công cụ mới miễn là không vi phạm bảo mật. Mặt hạn chế là Tabby (và AI nội bộ nói chung) **cần đầu tư thêm để đạt hiệu quả ngang các dịch vụ thương mại**.

Định hướng phát triển giải pháp AI nội bộ (Tabby) tại MSB

Dựa trên kết quả bước đầu, MSB xác định việc **tiếp tục phát triển giải pháp AI coding nội bộ** là cần thiết để nâng cao hiệu quả SDLC nhưng vẫn đảm bảo an toàn. Một số định hướng đề xuất cho Tabby trong thời gian tới:

- **Mở rộng phạm vi sử dụng**: Hiện Tabby mới pilot ở 3 dự án, kế hoạch là mở rộng ra **toàn bộ các đội phát triển ứng dụng kênh số** (Internet Banking, Mobile, API services). Đây là những lĩnh vực dùng ngôn ngữ hiện đại (Java, JavaScript, .NET) mà Tabby hỗ trợ tốt. Trước mắt, có thể triển khai thêm cho 5-7 dự án, khoảng 50 lập trình viên trong 2 quý tới. Việc này cần đi kèm đào tạo ngắn để dev biết cách tận dụng Tabby hiệu quả (ví dụ viết comment để gợi ý hàm, dùng chat hỏi lỗi...).
- **Nâng cấp mô hình và hạ tầng**: Để cải thiện chất lượng gợi ý, đội AI nên thử nghiệm các mô hình mã nguồn mở mạnh hơn. Ví dụ, **CodeLlama-34B** có thể được thay bằng **CodeLlama-34B-Instruct** hoặc **StarCoderPlus**, thậm chí nếu hạ tầng cho phép có thể thử **Llama-70B** (đã fine-tune cho code). Song song, cần tối ưu server (thêm GPU, tối ưu pipeline) để giảm độ trễ. Việc caching kết quả gợi ý cũng có thể nghiên cứu (Tabby có cơ chế caching theo Tree-sitter) nhằm phục vụ nhiều người dùng hơn. Mục tiêu là Tabby phản hồi < 1 giây và gợi ý "thông minh" hơn, tiệm cận trải nghiệm Copilot.
- **Tích hợp nguồn kiến thức nội bộ**: Một hướng rất tiềm năng là dùng Tabby như **trợ lý hiểu biết tài liệu nội bộ**. Với tính năng "Answer Engine" mới của Tabbygithub.com, MSB có thể nạp các **repository tài liệu thiết kế, hướng dẫn coding standard, tài liệu kiến trúc hệ thống** vào để AI tra cứu. Khi đó lập trình viên không chỉ được gợi ý code

mà còn có thể hỏi ngay trong IDE về **quy định coding của ngân hàng, cách gọi service nội bộ, v.v.**Điều này biến Tabby thành một **"wiki sống"** cho dev, giúp giảm thời gian tìm tài liệu và tuân thủ chuẩn tốt hơn. Để làm được, cần chọn lọc và chuyển định dạng các tài liệu kỹ thuật của MSB vào kho kiến thức cho AI và thiết lập cơ chế cập nhật thường xuyên.

- **Chính sách phân loại dữ liệu code**: Về mặt quản trị, ngân hàng có thể xây dựng **chính sách phân loại mã nguồn**theo mức độ nhạy cảm để hỗ trợ chiến lược AI. Code thuộc vùng nhạy cảm cao (core banking, dữ liệu KH…) sẽ **chỉ dùng AI nội bộ** như Tabby. Code thuộc vùng ít nhạy cảm hơn (ví dụ mã dịch vụ tiện ích, giao diện frontend, code mẫu từ open-source) có thể **được phép dùng AI bên ngoài** trong một số trường hợp. Việc phân loại này giúp tận dụng sức mạnh của các AI cloud khi cần thiết mà vẫn bảo vệ phần quan trọng. Tất nhiên, mọi trường hợp dùng AI cloud phải tuân thủ quy trình: code không chứa thông tin khách hàng hoặc bí mật kinh doanh, và phải được **duyệt bởi quản lý**. Chính sách rõ ràng sẽ giúp dev hiểu khi nào có thể dùng Copilot/Gemini một cách an toàn.

Đề xuất thử nghiệm các công cụ AI lập trình khác (PoC)

Mặc dù Tabby là giải pháp chính thức vì yếu tố bảo mật, MSB không nên bỏ qua **lợi ích từ các công cụ AI thương mại tiên tiến**. Đề xuất thực hiện một số **Proof of Concept (PoC)** nhỏ để đánh giá các công cụ đã nghiên cứu ở trên, từ đó có thêm phương án kết hợp hiệu quả:

- **GitHub Copilot (Microsoft)** – *Đề xuất:* Cho phép một nhóm giới hạn (5-10 dev) dùng Copilot trên một dự án *không nhạy cảm* (ví dụ phần front-end website công khai) trong 1-2 tháng. Mục tiêu để so sánh chất lượng gợi ý với Tabby và đo hiệu suất. Thiết lập tài khoản Copilot for Business để đảm bảo **không lưu trữ lịch sử chat và không dùng code cho huấn luyện**blog.gitguardian.com. Kết thúc PoC, thu thập phản hồi: Copilot có giúp code nhanh hơn nhiều không, có phát sinh vấn đề bảo mật (gợi ý mã license GPL chẳng hạn) không. Nếu kết quả tích cực, cân nhắc **mở rộng Copilot** cho các nhóm làm sản phẩm web công khai – nơi mà code không quá nhạy cảm và việc rò rỉ (nếu có) ít ảnh hưởng. Tuy nhiên, vẫn giữ nguyên tắc code review cẩn thận với mọi gợi ý từ Copilotblog.gitguardian.com để tránh lỗi ẩn.
- **Google Gemini Code Assist** – *Đề xuất:* Thử nghiệm cá nhân hoặc nhóm nhỏ trong phạm vi R&D: do công cụ này miễn phí, có thể giao cho vài lập trình viên thử dùng trên dự án *mẫu* (ví dụ viết một ứng dụng demo) để đánh giá khả năng. Đặc biệt chú ý kiểm tra tính năng **tự review code, giải thích code** của Gemini để xem có hữu ích cho việc **đào tạo nhân viên mới** không (vì một AI giải thích code rõ ràng sẽ giúp junior học nhanh hơn). Nếu Gemini tỏ ra xuất sắc và Google có lộ trình hỗ trợ doanh nghiệp (như cho triển khai on-prem hoặc hợp đồng bảo mật), MSB có thể cân nhắc sử dụng rộng hơn. Hiện tại, coi Gemini như một cách để **so sánh công nghệ** với Copilot/Tabby là chính.
- **AWS Kiro IDE** – *Đề xuất:* Phối hợp với một nhóm dự án mới (greenfield) dự kiến triển khai trên AWS để thử dùng Kiro ngay từ đầu. Ví dụ, một ý tưởng PoC là xây dựng nhanh một **ứng dụng web nội bộ nhỏ** (vd: dashboard báo cáo) dùng Kiro. Nhóm BA/Dev sẽ thử viết user stories cho Kiro, để AI sinh ra phần khung dự án. PoC này kiểm chứng xem **spec-driven AI** có thực sự tăng tốc phát triển không, và Kiro có dễ sử dụng cho dev hay không. Do Kiro tích hợp tốt với dịch vụ AWS, PoC nên chạy trên môi trường AWS

sandbox, không dính dữ liệu thật của ngân hàng. Nếu Kiro cho thấy khả năng xây dựng ứng dụng đúng yêu cầu trong thời gian ngắn (ví dụ giảm 30-40% thời gian so với bình thường), MSB có thể tính đến việc dùng Kiro cho các dự án phù hợp – nhất là dự án cần triển khai nhanh trên hạ tầng AWS. Tuy nhiên, yếu tố chi phí và bảo mật (Kiro gửi yêu cầu đến mô hình Claude của Anthropic) cũng phải đánh giá kỹ sau PoC.

- **LunaBase (Luna Studio)** – *Đề xuất:* Liên hệ với nhà cung cấp LunaBase để thử nghiệm bản **Enterprise on-premise** trong phạm vi giới hạn (nếu có chương trình trial). Chọn một nhóm dự án *full-stack* (cả frontend và backend) cỡ nhỏ để thử dùng Luna Studio trong 1 tháng. Mục tiêu đánh giá **khả năng đa tác vụ**: Luna có thực sự giúp code cả frontend + backend nhất quán không, các agent code review/test có bắt được lỗi hữu ích không. Đặc biệt, Luna hứa hẹn giảm 40% thời gian viết test và bắt ~95% bug trước khi đưa lên productionlunabase.ailunabase.ai – PoC sẽ đo xem có sát thực tế không. Nếu Luna chứng minh được hiệu quả và vấn đề triển khai on-prem ổn (có thể chạy trên server nội bộ, tích hợp LDAP...), MSB có thể cân nhắc mua license cho các **team phát triển sản phẩm mới** nơi yêu cầu cao về tốc độ ra mắt và chất lượng (ví dụ các dự án fintech nội bộ). LunaBase đắt nhưng nếu giúp **"10 người làm việc bằng 20 người"** thì vẫn xứng đáng. Ngược lại, nếu PoC cho thấy phức tạp, có thể dừng và chờ thêm sự trưởng thành của sản phẩm.
- **Các công cụ khác**: Ngoài danh sách trên, cũng nên theo dõi thêm những giải pháp AI coding khác đang nổi như **Cursor** (một VS Code fork có chat GPT-4), **CodeWhisperer của AWS** (tích hợp IDE, miễn phí cho cá nhân), **Codeium/Tabnine** (gợi ý code AI miễn phí/phí thấp)… Mặc dù chưa trong scope nghiên cứu chính, các công cụ này có thể phù hợp cho cá nhân dev dùng hỗ trợ ngoài giờ hoặc sử dụng trên mã nguồn công khai. MSB có thể khuyến khích văn hóa tìm hiểu các công cụ AI mới trong phạm vi an toàn – ví dụ cho phép lập trình viên đóng góp plugin AI mã nguồn mở vào IDE cá nhân miễn là không dùng với repo nội bộ. **Mục tiêu lâu dài** là xây dựng một **"AI-assisted development environment"** tại MSB, kết hợp hài hòa giữa giải pháp nội bộ (như Tabby) và các dịch vụ AI tiên tiến, nhưng dưới sự kiểm soát của ngân hàng.

Kết luận

AI cho phát triển phần mềm đã, đang và sẽ tiếp tục thay đổi cách lập trình viên làm việc. Từ việc gợi ý những dòng code đơn giản đến khả năng thiết kế cả kiến trúc hệ thống, AI hứa hẹn **tăng tốc độ phát triển, giảm lỗi và nâng cao hiệu suất toàn bộ vòng đời phần mềm**. MSB với đặc thù ngành tài chính cần tiếp cận xu hướng này một cách thận trọng nhưng không thể đứng ngoài cuộc. Việc triển khai thành công Tabby self-host cho thấy chúng ta có thể khai thác sức mạnh AI **mà vẫn đảm bảo an toàn thông tin**. Trong thời gian tới, bằng cách kết hợp các giải pháp AI tiên tiến một cách chọn lọc – ví dụ sử dụng AI nội bộ cho phần lõi nhạy cảm, dùng AI đám mây cho phần ít quan trọng hơn – MSB có thể xây dựng một quy trình phát triển phần mềm thông minh, hiệu quả. Quan trọng hơn cả, chúng ta cần trang bị cho đội ngũ **kỹ năng và tư duy mới** (tư duy đặc tả, kỹ năng kiểm soát đầu ra AI) để tận dụng AI như một "đồng đội" thực thụ. Nếu làm được điều đó, AI sẽ trở thành bệ phóng giúp MSB rút ngắn thời gian đưa sản phẩm số ra thị trường, nâng cao chất lượng và trải nghiệm người dùng – một lợi thế cạnh tranh lớn trong kỷ nguyên số hóa.

Nguồn tài liệu tham khảo: Các thông tin và số liệu trong nghiên cứu được tổng hợp từ nhiều nguồn uy tín, bao gồm báo cáo và blog công nghệ (GitHub, Google, AWS...), tài liệu nội bộ thí điểm tại MSB, cùng ý kiến chuyên gia về xu hướng AI trong lập trìnhsecondtalent.comsecondtalent.comnal.vnblog.gitguardian.com... Những nguồn này đã được trích dẫn trực tiếp trong nội dung để đảm bảo tính xác thực và minh bạch.