/thoughtworks

How Much Faster Can Coding Assistants Really Make Software Delivery?

Sichu Zhang
Apr 2025

How do you infuse GenAI into this?

- Market Research
- Competitor Analysis
- User Needs Identification
- Feasibility Study
- Technology Exploration
- Regulatory Requirements Identification
- Security Considerations
- Existing System Analysis (for modernization)
- Mobile Platform Analysis (for mobile projects)
- Data Analysis (for data projects)

- Project Scope Definition
- Communication Planning
- Project Governance
- Documentation Plan
- Mobile Platform Selection
- Resource Planning
- Budget Planning
- Risk Management
- Quality Assurance Planning
- Data Migration Plan
- Security Planning
- Project Scheduling

- Software Architecture Analysis
- Data protection impact assessment
- Requirement Gathering
- Compliance Requirement Identification
- Requirement Analysis
- Business Process Analysis
- User Interface Analysis
- Security Analysis
- Existing System Analysis
- Data Analysis
- Vulnerability Assessment
- Data Source Analysis

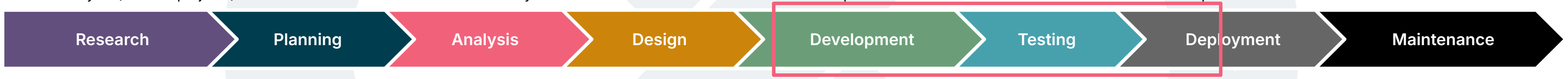
- System Design
- Software Architecture Design
- UI/UX Design
- Compliance-Driven Design
- Security Design
- API Design (for application development)
- Microservice Design (for modern applications)
- Mobile Design (for mobile applications)
- Data Pipeline Design (for data platforms)
- Risk management policy creation
- Incident response plan creation
- Infrastructure Design
- Database Design
- Privacy policy creation
- Terms of service creation
- Security policy creation
- Data retention policy creation

- Refactoring
- **Coding**
- Database Management
- UI/UX Implementation
- Security Implementation
- Reverse engineering
- Data Modeling
- Data Integration
- Data Security Implementation
- Integration
- API Development
- Testing (Unit and Component)
- Data Processing Implementation
- Continuous Integration
- Database Design
- Data Migration
- Data Validation
- Database Implementation

- Alpha Testing
- Compliance Testing
- Accessibility Compliance
- Defect resolution
- System Testing
- Non-functional Testing
- User Acceptance Testing (UAT)
- Beta Testing
- Mobile App Testing (for mobile projects)
- Data Testing (for data projects)
- Accessibility Testing
- Load and Stress Testing
- Compatibility Testing
- Defect tracking
- Unit Testing
- Integration Testing
- Functional Testing
- Regression Testing
- Automation Testing
- Test data management
- Test reporting

- Change Management
- Cutover Management
- Performance Tuning
- Mobile App Store Submission
- Environment Setup
- Release Preparation
- Data Migration
- Software Installation
- System Monitoring
- Load Balancing
- Compliance monitoring
- Backup and Recovery Setup
- SEO Setup

- Release and patch management
- System audits
- Incident Management
- Security Updates
- Performance Tuning
- Backup and Recovery
- System Documentation
- Compliance Checks
- Decommissioning
- Mobile App Updates
- Data Validation
- Security monitoring
- Performance monitoring
- Security monitoring
- Capacity planning
- Software Upgrades
- User Support
- Performance monitoring
- Application monitoring



Coding assistants probably only increase delivery speed by 10-15%, not the hyped 50%. This is still significant and cost-effective, but some argue that the focus on speed ignores a potential 41% increase in bug rates.

/thoughtworks

How Much Faster Can Coding Assistants Really Make Software Delivery?

GenAI has clear potential to increase speed and productivity.

Developers see the benefits

Industry

StackOverflow 2023 Developer Survey: "77% feel favorable towards using AI tools in their development workflow"

GitHub survey: "70% of developers see a benefit to using AI coding tools at work"

Thoughtworks

We're using coding assistance at **40 clients** and growing; over **2000** of our developers have had exposure to the tools.

Our internal surveys show clear positive feedback for usefulness. The internal **NPS of GitHub Copilot is over 30** (above 20 is very good)

But the hype is also leading to misguided expectations

What is the impact on team speed?

Getting the numbers straight - GitHub claims about Copilot:

- 35% acceptance rates of suggestions
- 40-60% portions of codebases written by Copilot (depending on language)
- Some *tasks* can be completed up to 55% faster.
- A GitHub developer survey says, "Teams spend 32% of their time coding"

What does that mean for the potential impact on story cycle time?

Scenario / assumptions	Part of cycle time spent on coding *	Part of coding supportable with Copilot	Rate of faster task completion with Copilot ˣ	Potential for time saved *in cycle time*
Very optimistic	40%	60%	55%	**13%**
Middle	30%	50%	45%	**7%**
Pessimistic	20%	40%	25%	**2%**

Depending on your cost structure, GitHub Copilot would make up around 0.01% of your team costs.

You need to look beyond speed.

- **Developer experience** improvements alone are worth it, and will improve onboarding and learning times.
- We've seen other positive impact, e.g. developers writing more tests in the time saved.

You need to look beyond coding.

- Coding is just one part of the software creation process.
- Using GenAI for things like requirements and testing analysis holds additional potential to improve quality of the delivery process and reduce waste.

Coding assistant impact on team speed at two clients

- Speed is the most asked for factor of team productivity when it comes to coding assistant impact.
- Cycle time is the most frequently used proxy variable for software team speed, it measures time from starting development work on a requirement to deployment.
- Rate of faster task completion is how much faster an **individual developer** can get when coding (GitHub's claim last year was 55%), whereas cycle time represents the whole team, including some of the non-coding work
- Part of cycle time spent on coding is a factor that **represents overall friction** and effectiveness of a team, the things we usually tackle with our **engineering effectiveness practices**. The higher a teams effectiveness is, the higher the *relative* impact of coding assistants on their speed can be. This heuristic helps us illustrate that to clients.

	Estimates			
Scenario	**Part of cycle time spent on coding**	**Part of coding supportable with coding assistant**	**Rate of faster task completion with coding assistant**	**Estimated time saved in cycle time**
Heuristic				
Optimistic	40%	60%	55%	**13%**
Client 1 - Team manually tracked their estimated savings across ~150 tickets Tool: GitHub Copilot with GPT models				
	55%	50%	30%	**8%**
Client 2 - Some teams are able to use the tool for all of their tasks (suitable prevalent tech stack) Tool: Cline with Claude Sonnet model				
VERY HIGH - Teams with low friction*	60%	100%	80%	**48%**
HIGH - Teams with higher friction*	40%	100%	80%	**32%**
MEDIUM - Upgrading a complex existing rules engine	40%	100%	40%	**16%**

**friction = delivery friction like dependencies, inefficient deployment processes, etc*

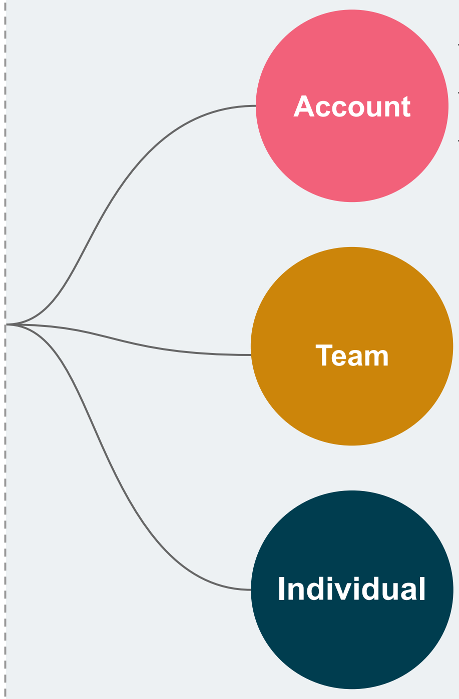
Problem statement for launching an AI Capability Growth

Client expectation

- ❖ Deploy OOTB (Out of the Box) AI enablement capabilities internally to drive efficiency and effectiveness across all of organization.
- ❖ There is absolutely no excuse for not making AI assisted development part of your workflow. With all the productivity evidence and free assistance and training provided, **I would not even consider engineers for a job without skills and experience using AI dev tools.** Invest time in your career.

Industry Trend

- ❖ We believe there is an opportunity for TWs delivery teams to leverage AI tooling to help us deliver more efficiently to assist our clients in getting value to market faster.

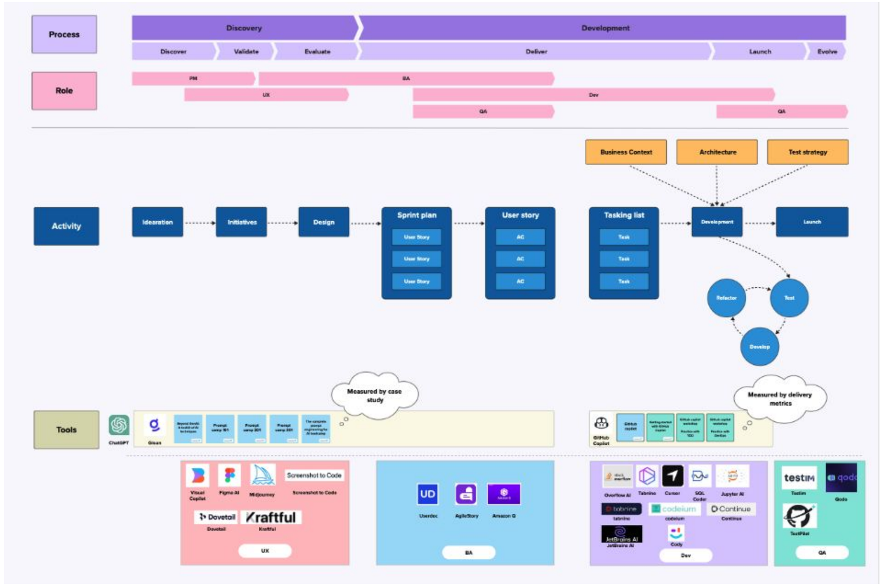


- ❖ #1: No clear and achievable expectations for the use of AI tools
- ❖ #2: No clear guidance on how to measure capability growth
- ❖ #3: No clear measure of success from the use of AI

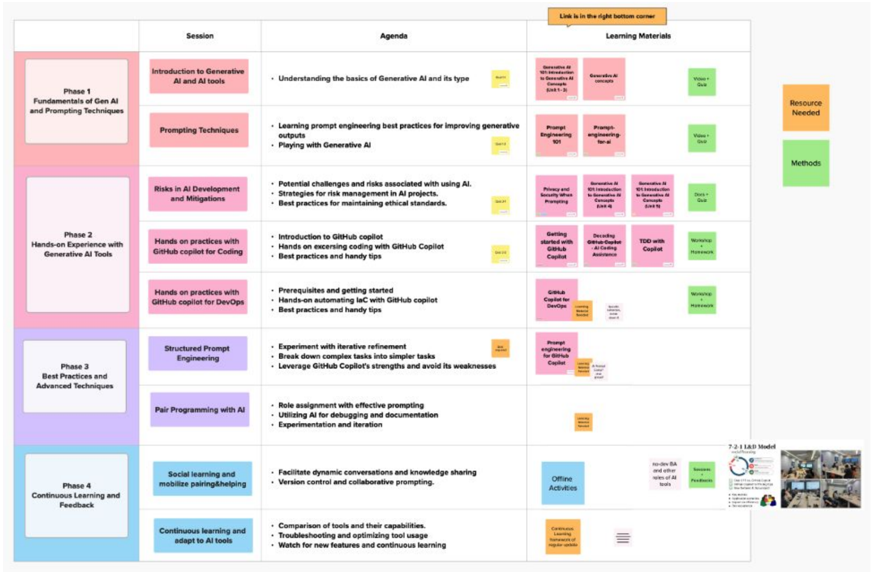
- ❖ #4: Not yet able to hands on development
- ❖ #5: Not sure how to achieve best practices in a team context

- ❖ #6: AI Development is progressing very rapidly and it can be confusing for individual to figure out what to use when
- ❖ #7: Hard to follow the current trends

AI Capability Growth Framework



Building AI Capability Growth framework based on Discovery-and-Delivery-Process to solve #1, #3, #4, #5, #7. It allows us to look at the integration of AI in each activity from an agile engineering perspective, and the potential tooling options also allow us to keep our eyes on development trends.


Link is in the right bottom corner

	Session	Agenda	Learning Materials
Phase 1 Fundamentals of Gen AI and Prompting Techniques	Introduction to Generative AI and AI tools	• Understanding the basics of Generative AI and its type	Generative AI 101: Introduction to Generative AI Concepts (Unit 1 - 3); Generative AI concepts; Video + Quiz
	Prompting Techniques	• Learning prompt engineering best practices for improving generative outputs • Playing with Generative AI	Prompt Engineering 101; Prompt-engineering-for-ai; Video + Quiz
Phase 2 Hands-on Experience with Generative AI Tools	Risks in AI Development and Mitigations	• Potential challenges and risks associated with using AI. • Strategies for risk management in AI projects. • Best practices for maintaining ethical standards.	Privacy and Security When Prompting; Generative AI 101: Introduction to Generative AI Concepts (Unit 4); Generative AI 101: Introduction to Generative AI Concepts (Unit 5); Docs + Quiz
	Hands on practices with GitHub copilot for Coding	• Introduction to GitHub copilot • Hands on excersing coding with GitHub Copilot • Best practices and handy tips	Getting started with GitHub Copilot; Decoding ~~GitHub Copilot~~ - AI Coding Assistance; TDD with Copilot; Workshop + Homework
	Hands on practices with GitHub copilot for DevOps	• Prerequisites and getting started • Hands-on automating IaC with GitHub copilot • Best practices and handy tips	GitHub Copilot for DevOps; Learning Material Needed; Workshop + Homework
Phase 3 Best Practices and Advanced Techniques	Structured Prompt Engineering	• Experiment with iterative refinement • Break down complex tasks into simpler tasks • Leverage GitHub Copilot's strengths and avoid its weaknesses	Prompt engineering for GitHub Copilot; Learning Material Needed
	Pair Programming with AI	• Role assignment with effective prompting • Utilizing AI for debugging and documentation • Experimentation and iteration	Learning Material Needed
Phase 4 Continuous Learning and Feedback	Social learning and mobilize pairing&helping	• Facilitate dynamic conversations and knowledge sharing • Version control and collaborative prompting.	Offline Activities; no-dev BA and other roles of AI tools; Sessions + Feedbacks
	Continuous learning and adapt to AI tools	• Comparison of tools and their capabilities. • Troubleshooting and optimizing tool usage • Watch for new features and continuous learning	Continuous Learning framework of regular update

Resource Needed

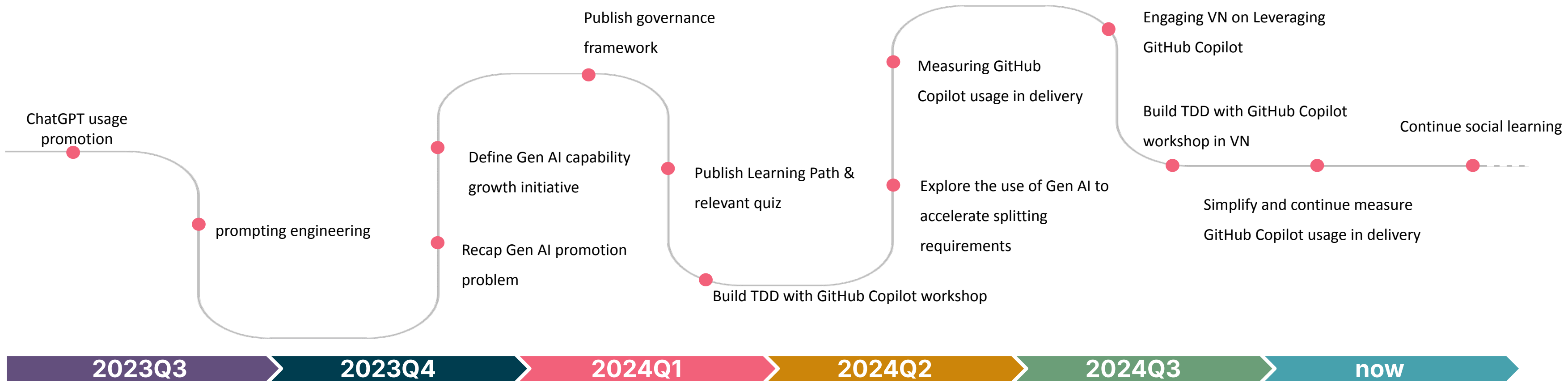
Methods

7-2-1 L&D Model

From a practical perspective, a learning path map is developed to enable individuals to learn the fundamentals, hands-on practice and trendy development directions end-to-end to address #2, #5, #6, #7

AI Capability Growth Roadmap

Our Journey of AI Capability Growth Initiative



ChatGPT usage promotion
prompting engineering
Define Gen AI capability growth initiative
Recap Gen AI promotion problem
Publish governance framework
Publish Learning Path & relevant quiz
Build TDD with GitHub Copilot workshop
Measuring GitHub Copilot usage in delivery
Explore the use of Gen AI to accelerate splitting requirements
Engaging VN on Leveraging GitHub Copilot
Build TDD with GitHub Copilot workshop in VN
Simplify and continue measure GitHub Copilot usage in delivery
Continue social learning
2023Q3
2023Q4
2024Q1
2024Q2
2024Q3
now

> 90%

Finish 7 sets of quizzes for Learning path stage 1~4

80.5%(Xian)
61.9% (VN)

devs completed the TDD with Github Copilot workshop

100%

- ❖ **Everyone who participates in the workshop submit at least one insight, including ticket, commit, etc**
- ❖ **Keep collecting insights from new member by team iteratively collecting GitHub Copilot usage data**

15.8%

Using GitHub Copilot reduced coding time for four major usage scenarios, reducing the average cycle time of each team by 15.8%

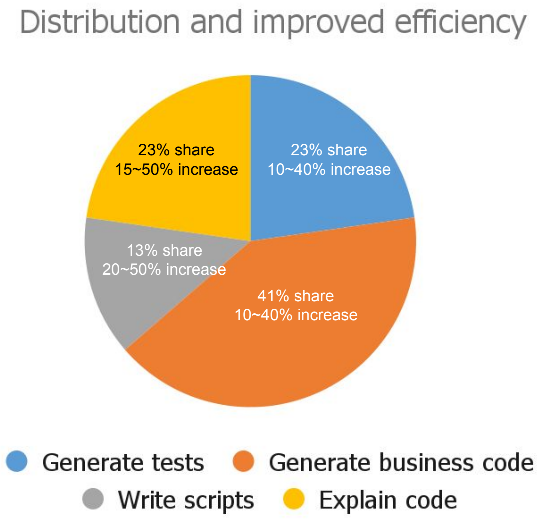
The team tracked 150 tickets over time and documented task type, AI Assistants usage and estimated time saved

Ticket	AI Assistant used	Reason for (non-)usage	Estimated time saved
XXX-345	Yes	Generated business code	30%
XXX-362	No	Bug fix for a known issue that only needs a tiny code change	n/a
XXX-385	Yes	Generated code and test data	40%
XXX-347	Yes	Generated shell scripts	50%
XXX-312	No	Spike with lots of research	n/a

AI Assistants Applicability Analytics

Getting feedback from all team members that participated in the TDD with AI Assistants workshop. Picking up 1 of sprint data, these teams completed a total of 150 tickets in the recent iteration. Among them, 73 tickets did use AI Assistants (48.7%), 77 tickets did not use AI Assistants (51.3%).

The teams' usage scenarios can be roughly divided into four categories:



- ❖ Generate test: including generating test code and test data. Because the code structure is simple, the accuracy is higher.
- ❖ Generate business code: the accuracy is limited by the business context, and the probability of the generated code being adjusted is relatively high, about 30% ~ 80%.
- ❖ Write scripts: Since the generated scripts are less relevant to the business, the prompting needs to contain less information and the generated code is more accurate.
- ❖ Explain code: Teams use code explanation as a more efficient way to collect business code and information, and some teams even use it to share business context in a session, especially for the legacy code.

Scenarios where AI Assistants is Not used:

- ❖ Local environment setup
- ❖ Manual tests
- ❖ Knowledge sharing
- ❖ Straightforward tasks
- ❖ **Tools decommission or upgrade**
- ❖ Designing and Solutioning
- ❖ **Alerting & monitoring**
- ❖ Supporting requests
- ❖ **Vulnerability fixing**
- ❖ Troubleshooting cross the system

AI Assistants Increases Speed and Productivity

Scenarios	Development time	Generate tests		Generate business code		Write scripts		Explain code		Applicable weighted cycle time saved **[S]**	General weighted cycle time saved **48.7%**
		23% **[Zi]**		41%		13%		23%			
		Coding time saved	Cycle time saved	Coding time saved	Cycle time saved	Coding time saved	Cycle time saved	Coding time saved	Cycle time saved		
Very optimistic	76.5% **[X]**	30.0% **[Yi]**	23.0%	50.0%	38.3%	50.0%	38.3%	40.0%	30.6%	**33.0%**	**16.1%**
Middle	55.4%	28.0%	15.5%	29.4%	16.3%	38.7%	21.4%	22.0%	12.2%	**15.8%**	**7.7%**
Pessimistic	48.0%	15.0%	7.2%	10.0%	4.8%	33.3%	16.0%	10.0%	4.8%	**6.8%**	**3.3%**

$$S = \sum_{i=1}^{n} X \times Y_i \times Z_i \quad n \in \{1,2,3,4\}$$

48.7%: According to the most recent iteration, the team has completed about 150 tickets in total. Of these, 73 tickets did use AI Assistants (48.7%).

X = Development Time *(the percentage of time the team spends on implementation and testing during a sprint. In the table above, this value represents the average for the Xi'an team in each segment over the past two months)*

Y = Coding Time saved *(the percentage of coding time saved based on the analysis of each ticket collected by the team capability champion)*

Z = Usage Scenario Proportion *(a statistical value representing the proportion of tickets for this type of usage scenario in the sprint)*

S = Applicable Cycle Time Saved *(the average improvement rate of overall cycle time across teams after weighting each usage scenario (currently 4 usage scenarios))*

Smart Alert Diagnosis and Troubleshooting with AI

Integrating AI monitoring tools to reduce alert handling time and enhance service stability.

An Australian multinational software company offering accounting solutions for SME's wanting to reduce issue identification time and enhance service stability, developed an automated AI analysis tools, integrating various monitoring metrics, health checks, Sumo logic metrics to automatically obtain the current system status and indicators, and integrated automated responses in Slack. This information will is being integrated with Glean to generate final troubleshooting steps based on existing documentation and providing recommendations via Slack.

This allows to reduce manual effort and quickly identify and resolve production issues for all 13 systems or services, reducing issue identification time from 40 minutes to under 15 minutes and achieving a 99.97% system availability.

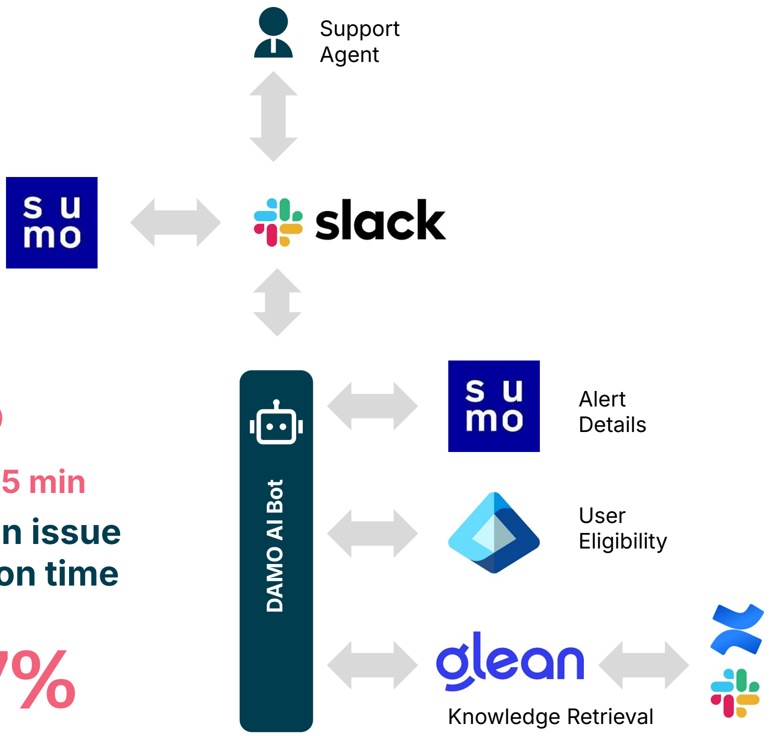
-25%

From 30 to 15 min

Reduction in issue identification time

99.97%

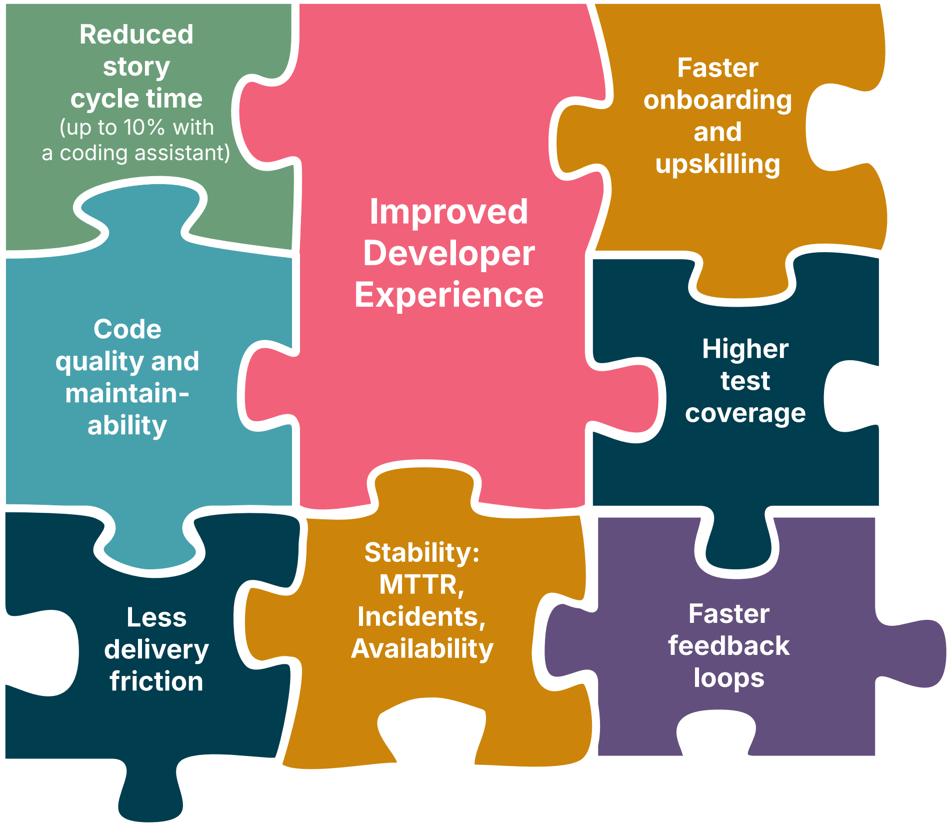
Maintained High System availability



How much AI will save you on productivity?

There is no "X%" answer to this question.

How to measure developer productivity is one of the **constant challenges** in software delivery, and it's multidimensional. This doesn't change with the advent of AI tools.



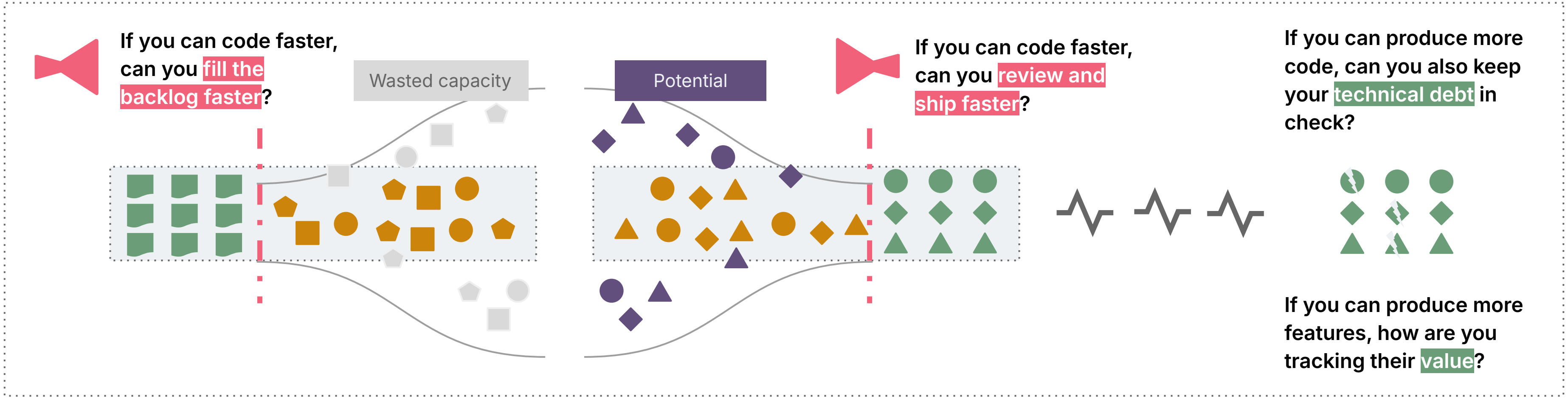
Reduced story cycle time (up to 10% with a coding assistant)
Improved Developer Experience
Faster onboarding and upskilling
Code quality and maintain-ability
Higher test coverage
Less delivery friction
Stability: MTTR, Incidents, Availability
Faster feedback loops

Engineering practices still matter, if not more.

Good practices help deal with bottlenecks from higher coding throughput.

GenAI amplifies your status quo - the good gets even better, but the bad can get worse.

Good practices mitigate the GenAI risks, and help manage the quality of more code.


If you can code faster, can you fill the backlog faster?
Wasted capacity
Potential
If you can code faster, can you review and ship faster?
If you can produce more code, can you also keep your technical debt in check?
If you can produce more features, how are you tracking their value?

Common impediments to flow & their cost

Examples of common friction that are draining productivity and block flow

Finding information - The usually "hidden" cost of not quickly finding, or not finding at all, important knowledge about APIs, integrations & systems.

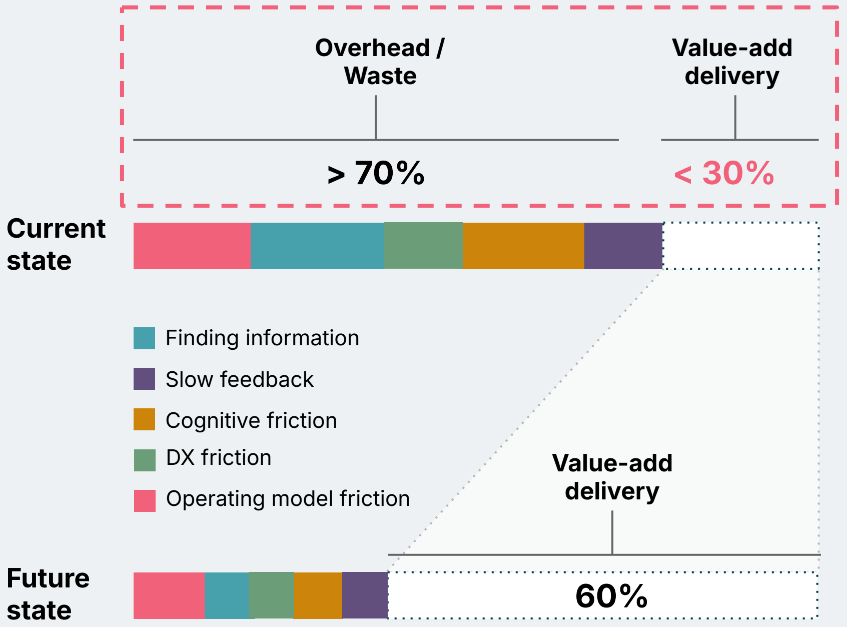
DevEX friction - The cost of productionizing software so that it is highly available. Includes drag from deployment, testing, observability, and resiliency. Also, the cost of having to use clunky tools, poorly designed APIs, lack of self-service capabilities.

Cognitive overload / task switching - The cost of misunderstood integrations, abstractions, and data. Cognitive taxes create quality issues, slowing delivery significantly.

Slow quality feedback loops - Defects caught in pre-prod or prod have exponentially higher cost to remediate and disrupt the flow of value added work.

Operating model friction - Lost time and rework due to the flow of work between product and engineering, such as poorly specified product specs and architecture requirements and review.

Team effectiveness



Q&A

thoughtworks