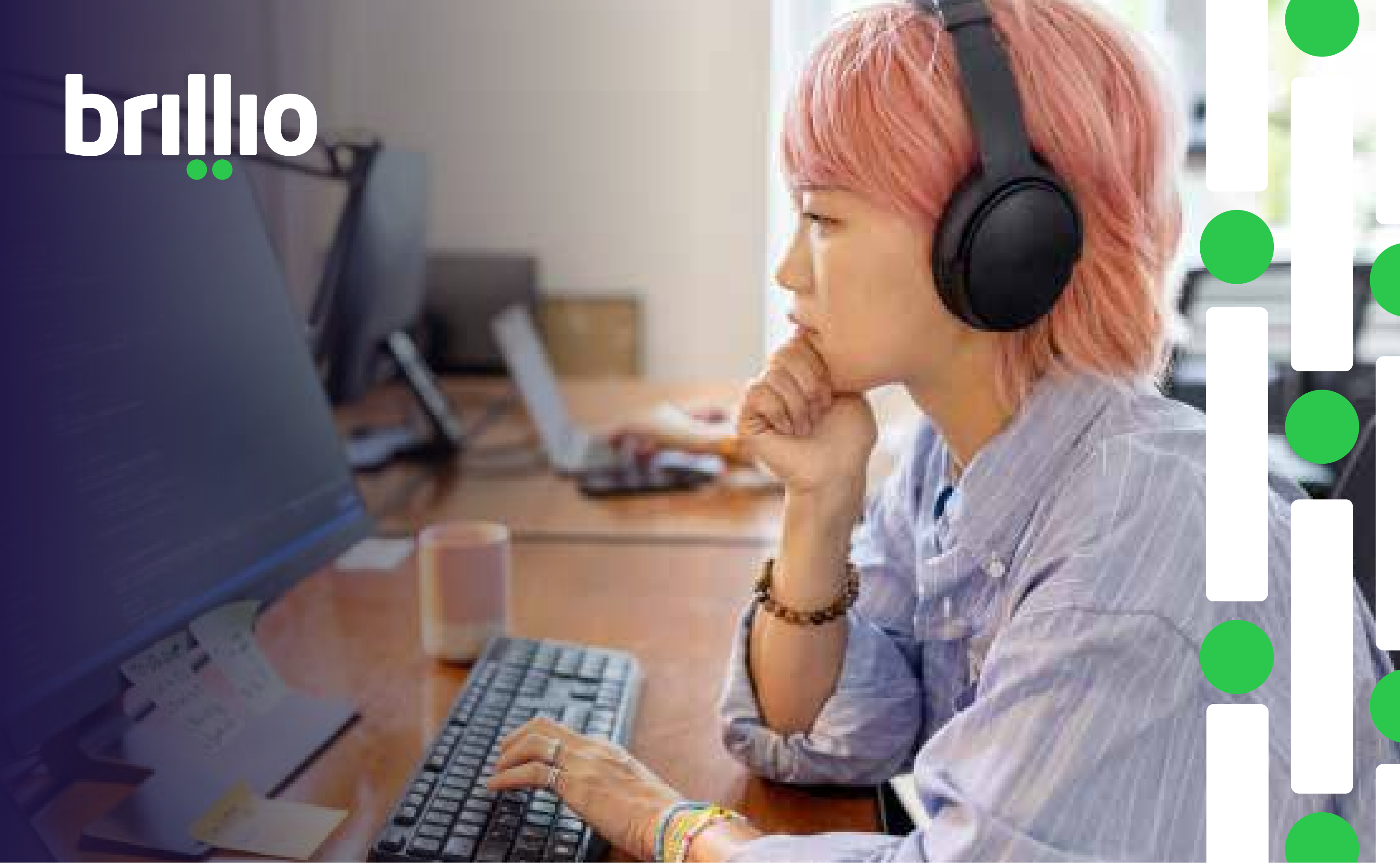


Simplify complex functional programming tasks with AI

GenAI can streamline functional programming (FP) with automation, enabling developers to focus on the creative and intricate aspects of platform engineering

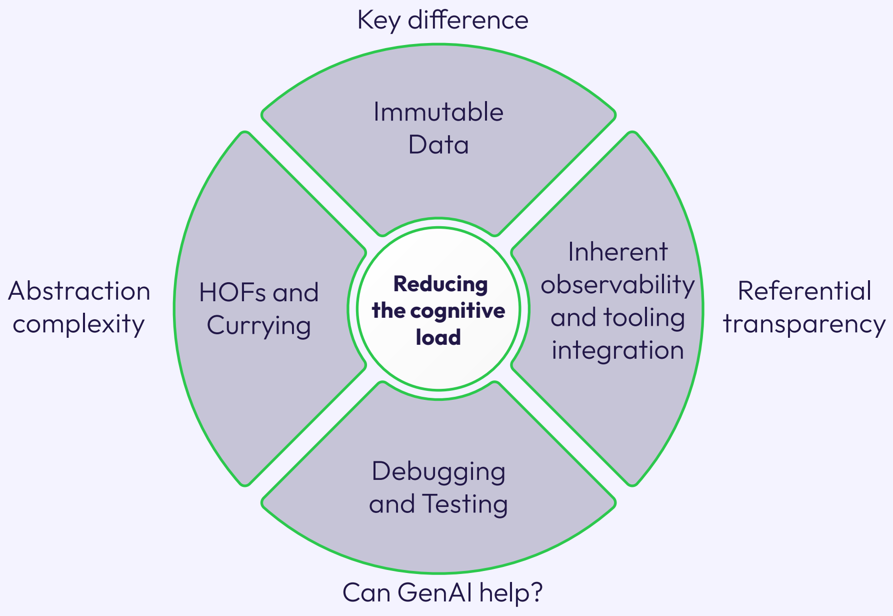
Key aspects that enhance an **FP developer's experience**

What are the key aspects that make life easier for FP developers? First, immutability helps reduce bugs and avoid race conditions, making concurrent programming safer. Then, there's modularity, where one can rapidly reuse and scale components. Higher-order functions let developers pass functions around and create powerful abstractions. Declarative programming makes code readable and easier to access. Lazy evaluation boosts performance by only evaluating expressions when needed and even supports infinite data structures. Finally, robustness ensures the code is more resilient and has fewer bugs, thanks to static typing that catches errors early. Together, these aspects make for a smooth and reliable development process.

Key challenges that hurt an **FP developer's experience**

FP developers face several challenges too. These include a steep learning curve with new concepts and methods, performance issues stemming from high memory usage, the need to avoid mutable data, and missing imperative features like database operations and verbosity. Additionally, there are concerns about resource overheads due to recursion and higher CPU usage, a limited tooling ecosystem, and industry adoption issues, as functional programming is not a mainstream field and requires skilled programmers.

Developer experience challenges in **functional programming** are not unique, but there are some differences from **imperative programming**.



Why are these **important?**

The **'value'** of time a human spends at most software companies is **20–100x** the value of machine time because of human expertise in decision-making and high costs of automation. Optimizing capabilities must always be done to save developer time and effort. The technology landscape evolves rapidly and instilling a robust developer experience is key to driving success in the modern software development lifecycle. Maintaining a high degree of clarity in optimizing the developer experience yields several benefits. Happy developers are more likely to be productive and innovate, reducing cognitive load and leading to faster and more reliable software releases. They collaborate better and stay updated with the latest technology offerings, making them more adaptable to change management.

On the contrary, a not-so-great developer experience can lead to friction, impeding development cycles, turnover, and product quality. The outcome of a positive developer experience is more than just about having good metrics, it's about delivering business value. If one waits for the metrics to determine if there is a problem, then it's too late. Developer experience friction takes shape in a few ways, as follows:

Developer experience friction takes many shapes

People

- Understand pain and friction points
- Understand skillset challenges
- Understand organizational structure friction

+

Process

- Be metrics-driven
- Understand that all friction is not tech-based
- Value stream map to identify lead, wait, and overall cycle times

+

(Too much focus here)

Technology

- Right tools for the job
- Well-defined ownership
- Continuous improvement based on feedback

People

Understand individuals' pain points and challenges, such as friction arising from skill gaps and organizational structure. Enterprises must be cognizant of these human factors to allay any concerns that may lead to conflict.

Process

To address processes, teams must focus on being metrics-driven but understand that all friction is not solely technology-based. Use tools like value stream mapping to assess lead and wait times and overall process cycle times.

Technology

Technology is another area that leads to developer experience friction and too much focus is placed on it as being the sole cause. While having the right tools and platforms is important, there must be more focus on ownership and continuous improvement based on feedback instead of relying on tech alone to solve every problem.

A deep dive into developer experience **and its significance**

Developer experience challenges in FP are not unique, but there are some stark differences from imperative programming as follows:

- **Abstraction complexity:** It can be hard to understand how different parts of the code fit together. Immutable data: Data cannot be changed once created, which can be tricky to manage.
- **Referential transparency:** Functions always produce the same output for the same input, which is different from how things usually work in imperative programming.
- **Observability and tooling:** It can be harder to monitor and debug functional programs because of their unique nature.
- **Debugging and testing:** Finding and fixing bugs can be more challenging.

GenAI can help address these challenges, offering solutions that make tasks easier by reducing the mental effort needed to work with functional programming concepts.

Developer experience **principles**

Principle 1: Keep it simple, stupid (K.I.S.S.)

Emphasize simplicity in software development. Maintain clear and concise documentation, practical examples, and clear error messages to aid developers. Furthermore, APIs and libraries should be intuitive, avoiding unnecessary complexity and abstraction. These considerations drive a good experience by making code more accessible and easier for developers to work with.

Principle 2: Consistency

Craft well-defined and documented standards across software development, including naming conventions, coding styles, and design patterns. Consistency improves readability, reducing errors and enhancing collaboration.

Principle 3: Fail fast

Strive to build and test code quickly to find and fix issues early. With fast build times, automated tests, and instant previews, developers can spot problems immediately and correct them before they snowball into bigger issues. This approach improves the overall quality and efficiency of the software development process.

Principle 4: Tooling ecosystem

Invest in an engineering platform that abstracts complexities, allowing for a modular and interchangeable self-service paradigm. This entails building a system where tools and components can be easily added, replaced, or customized without extensive manual intervention. Such platforms enhance flexibility and scalability, enabling developers to focus on innovation and problem-solving instead of being bogged down by infrastructure constraints.

Principle 5: Learning

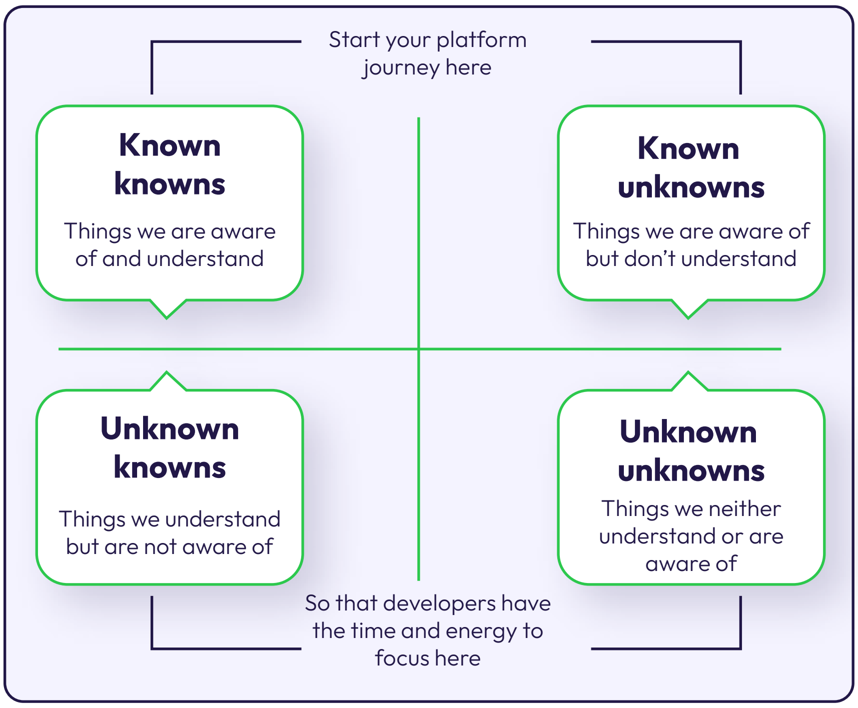
Lean on supervised learning and supportive models to enhance development processes. By leveraging these models, teams can effectively share knowledge, improve skills, and drive innovation, ensuring that learning is integral to their workflow. This collaborative learning environment fosters growth and adaptability, enabling teams to stay ahead in a rapidly evolving landscape.

Platform engineering **to the rescue**

Platform engineering provides composable and replaceable self-help capabilities. It can accelerate application delivery and the pace at which developers produce business value. By 2026, 80% of large software engineering organizations will establish platform engineering teams as internal providers of reusable services, components, and tools for application delivery. Platform engineering will ultimately solve the central problem of cooperation between software developers and operators (Gartner).

Starting the **platform journey**

Leverage your platform


Start your platform journey here
Known knowns
Things we are aware of and understand
Known unknowns
Things we are aware of but don't understand
Unknown knowns
Things we understand but are not aware of
Unknown unknowns
Things we neither understand or are aware of
So that developers have the time and energy to focus here

Let's look at the four quadrants above and how to leverage a platform effectively.

Known knowns

These include aspects we are both aware of and understand well. This is where the platform journey should start ideally, where clarity exists.

Known unknowns

These include things we are aware of but don't fully understand. A platform can help bridge this gap by providing insights that improve understanding.

Unknown knowns

This quadrant includes things we understand but who's existence we are not yet aware of. By leveraging a platform, developers can focus on surfacing and addressing hidden or underutilized areas.

Unknown unknowns

There are the most challenging in the quadrant—things that neither are understood nor recognized. The platform journey eventually aims to reduce uncertainty in this quadrant.

Start your platform journey from the areas of known information and understanding, gradually moving toward addressing knowledge gaps so developers can focus their time and energy on deeper unknowns and more valuable tasks.

GenAI’s role in **platform engineering**

GenAI plays an important role in platform engineering by automating and optimizing complex processes, enhancing developer productivity, and improving overall system reliability. Integrating GenAI into platform engineering allows developers to focus on innovation and higher-level tasks while AI handles predictive operations. GenAI’s ability to incrementally learn and improve will ultimately reduce errors, improve scalability, and accelerate software development lifecycles, reshaping how platforms can be engineered and maintained.

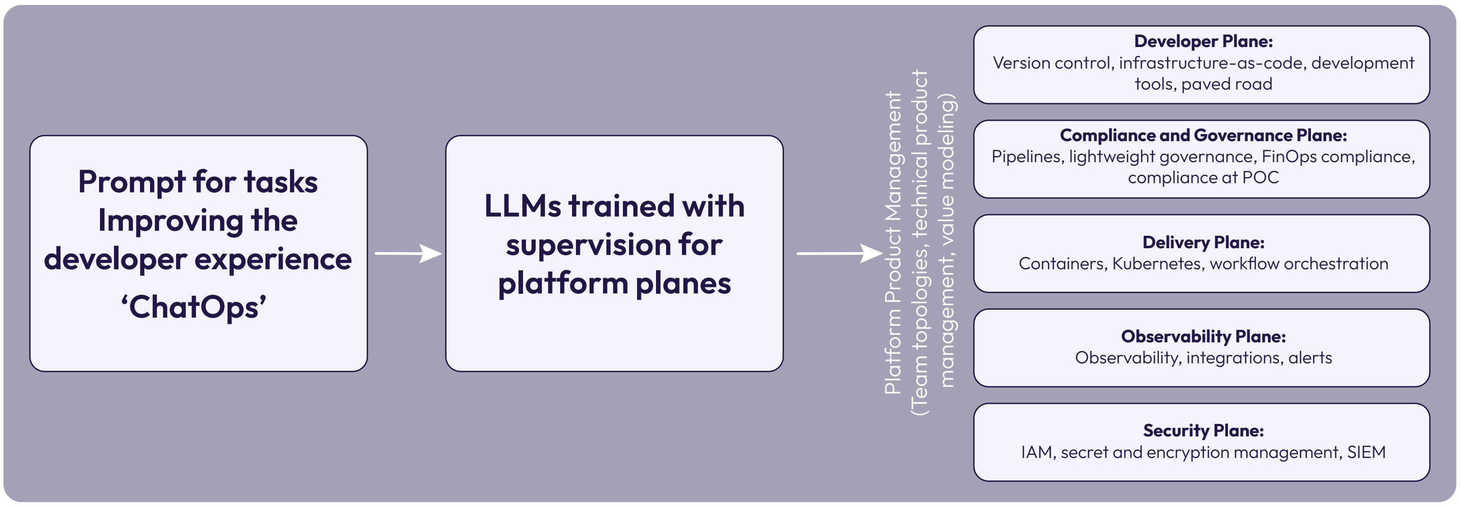
However, one of the challenges in leveraging GenAI is the occurrence of **hallucinations**—instances where AI generates information that seems plausible but is inaccurate or irrelevant. These hallucinations can disrupt processes, particularly in high-stakes platform environments where precision and reliability are key. To mitigate such disruption, performance gaps in GenAI tools are supplemented with Brillio’s **proprietary IP and accelerators**. These solutions ensure that when GenAI falls short, Brillio's enhancements provide the necessary accuracy and operational consistency, addressing any discrepancies that might arise from hallucinations or tool limitations.

GenAI can filter out valuable insights amidst the noise, focusing on critical signals that matter the most to performance, efficiency, and problem-solving. It continuously monitors data and flags when performance moves beyond predefined thresholds, indicating areas for developers to act. Developers can leverage AI to identify deviations, anomalies, or inefficiencies, ensuring focus on meaningful improvements while reducing distractions arising from non-essential events. Some popular AI code assistants for functional programming include Codeium, Tabnine, GitHub Copilot, Replit, and Blackbox AI. Some AI code review tools for functional programmers include Codacy, deepsource, Sonarqube, CodeScene, and GitHub Copilot.

In the 'build vs buy' debate, the **build** approach offers advantages in terms of customization and control since developing AI in-house will tailor solutions to meet business needs that align perfectly with the platform’s goals. However, it comes at a high cost, requiring significant financial investment and highly specialized talent. The **buy** approach, with pre-built tools like the ones mentioned earlier, offers speed and cost-effectiveness, allowing for quicker integration and benefits realization. However, to overcome the limitations of pre-built solutions, such as the potential lack of flexibility and control, Brillio enhances these platforms with its **accelerators** to provide superior customization and close the gap on any shortcomings that arise due to vendor dependencies or integration challenges.

Know what **'good' looks like**

How do we measure the **ROI** of **AI assistance**? Organizations must focus on how AI improves key performance indicators related to benefits from productivity, cost savings, time-to-market, developer satisfaction, and system reliability. Quantify these benefits against AI implementation costs to provide a clear picture of the AI'sfinancial and operations returns that will ultimately enhance the developer experience and the engineering process. Apply prompts and supervised training to improve both through AI-driven automation. Let's look at the key components below.



Prompt for tasks Improving the developer experience 'ChatOps'
LLMs trained with supervision for platform planes
Platform Product Management (Team topologies, technical product management, value modeling)
Developer Plane: Version control, infrastructure-as-code, development tools, paved road
Compliance and Governance Plane: Pipelines, lightweight governance, FinOps compliance, compliance at POC
Delivery Plane: Containers, Kubernetes, workflow orchestration
Observability Plane: Observability, integrations, alerts
Security Plane: IAM, secret and encryption management, SIEM

With ChatOps, developers can interact with their platform via chat-based commands to speed up task execution, aimed at improving the experience by streamlining routine actions. LLMs must be trained with supervision and tailored for platform planes to understand prompts and execute tasks based on platform requirements to ensure they align with platform-specific processes. Let's break down the different areas of platform engineering that an LLM would need to interact with.

- **Developer plane:** Deals with version control, infrastructure-as-a-code, developer tools, and roadmap planning.
- **Compliance and governance plane:** Ensures that pipelines, governance policies, and compliance standards (like FinOps) are followed.
- **Delivery plane:** Manages containerization, Kubernetes, and orchestration for automating workflows.
- **Observability plane:** Focuses on observability tools, integrations, and alerts for monitoring the system's health.
- **Security plane:** Covers identity and access management, encryption, security information, and event management.

Improve developer efficiency by automating tasks while ensuring compliance, security, and observability across the platform. A cohesive AI platform allows for a central point for the community to find new and existing capabilities across tooling, platforms, and AI capabilities. In a fast and changing landscape, having quick feedback loops and understanding the realization of business value are key to driving overall success. The AI portal must be part of the developer or platform strategy, not a standalone tool.

GenAI in platform engineering catalyzes a transformative shift for functional and imperative programmers alike. By automating the mundane and illuminating new possibilities, it empowers developers to innovate at the frontier of technology, redefining the boundaries of what can be built and how swiftly it can be brought to life.

About Brillio

Brillio is one of the fastest growing digital technology service providers and the partner of choice for many Fortune 1000 companies seeking to turn disruption into a competitive advantage through innovative digital adoption. We help clients harness the transformative potential of the four superpowers of technology: cloud computing, Internet of Things (IoT), artificial intelligence (AI) and mobility. Born digital in 2014, we apply our expertise in customer experience solutions, data analytics and AI, digital infrastructure and security, and platform and product engineering to help clients quickly innovate for growth, create digital products, build service platforms, and drive smarter, data-driven performance. With 17 locations across the U.S., the UK, Romania, Canada, Mexico, and India, our growing global workforce of nearly 6,000 Brillians blends the latest technology and design thinking with digital fluency to solve complex business problems and drive competitive differentiation for our clients. Brillio has been certified by Great Place to Work since 2021.

https://www.brillio.com/
Contact Us: info@brillio.com