cortex

Platform Engineering Interview Guide for Developers

Understanding your audience to build a better roadmap

This guide uses the Jobs, Pains, Gains framework and the Jobs-to-be-Done (JTBD) methodology to help platform engineers gather insights on the type of developer platform they should build. The focus is on identifying developer needs, pain points, and desired outcomes to inform the design and implementation of a new Internal Developer Platform.

Table of Contents

Section 1

Introduction & Context Setting

Objective: Set the stage for the interview by providing context and explaining how the feedback will be used.

1. **Introduction:**
    - "We’re looking to design a new internal developer platform tailored to your needs, so today we’ll discuss your workflow, challenges, and what would make your job easier."
2. **Set the agenda:**
    - "I’ll be asking about the tools you use, your development process, areas of frustration, and what features or services would help you work more effectively."
3. **Explain the outcome:**
    - "Your feedback will help us create a platform that improves your productivity, automates common tasks, and integrates the tools you need."

Section 2

Developer Jobs (Jobs-to-Be-Done)

Objective: Identify the tasks developers are trying to accomplish and how the platform can support their workflow.

1. **Core Development Objectives:**
    - “What outcomes are you expected to deliver in your role?”
    - “Which outcomes do you feel like you’ve been able to achieve with little friction? (i.e. you feel you’re doing well at this particular goal, and are happy with how much effort it requires for you to hit these goals)
    - Irrespective of what it would take to do so, which outcomes do you wish you were able to deliver more comprehensively, and/or faster?”

2. **Development Process:**

- "Walk me through your typical development cycle from coding to deployment. What are the major steps? Who do you interact with throughout?”

3. **Tool Preferences and Expectations:**

- "What tools are you using at each stage of development?"
- "What tools are non-negotiable for your workflow? Why do you prefer them?"

4. **Security and Compliance**

- "How do you ensure code and infrastructure comply with security standards? What would make this easier for you?"
- **Example tools:** Built-in security checks using tools like Snyk or Aqua Security, automated compliance scans in CI/CD pipelines.
- “To what extent are you responsible for security compliance for your app?”
- **Example responsibilities:** “I must audit my open source dependencies on each build.” or “I have to scan all bespoke code with CodeScene”

5. **Error Handling and Rollbacks:**

- "What’s your biggest concern when something goes wrong in production? How should the platform help you resolve it?"
- **Example features:** One-click rollbacks, integrated incident management tools like PagerDuty, detailed error logs.

Section 3

Pain Points, Challenges, Opportunities

Objective: Understand the pain points developers face in their current workflow to ensure the new platform addresses them.

1. **Onboarding:**

- “How did you find your onboarding experience? When did you feel like you had all the information, tools, and templates you needed to operate completely independently?”
- “Did anything in your onboarding unlock greater speed in gaining context?”

- “Did anything about your onboarding feel particularly painful or confusing?”

2. Current Bottlenecks:

- "Where do you encounter the most friction or delays in your workflow? Please answer irrespective of whether you believe there’s a solution to that problem.”
- “How has that changed over your time working here? Was there an “unlock” moment you can recall?”
- **Example issues:** Hard to find information, slow CI/CD pipelines, manual configuration of environments, troubleshooting deployment failures.

3. Time-Consuming Tasks:

- "What tasks feel like they take up more time than they should? Where do you feel you're wasting time?"
- **Example tasks:** Finding templates, reconfiguring build pipelines, waiting for secrets allocation, repetitive testing steps.

4. Tool Challenges:

- "Are there any tools you’re expected to use that you think are less useful, or maybe even slow you down?”
- "Do you struggle to integrate different tools in your current setup? If so, which ones?"
- **Example issues:** Jenkins not playing well with Docker, lack of visibility into Kubernetes clusters, or difficulties connecting monitoring tools like Prometheus and Grafana.

5. Collaboration and Communication Barriers:

- "How easy or difficult is it to collaborate with other teams? Does the current tooling support cross-team collaboration effectively?"
- **Example issues:** Disjointed communication between Ops and Dev teams, unclear ownership of services, missing documentation.

6. Work Satisfaction

- “How satisfying is it to do your work on a daily basis?”
- “What are some improvements that could be made to help your work become more satisfying or fulfilling?”
- **Example responses:** “I wish that there were less approval gates.” or “I have too many outbound dependencies.”

Section 4

Ideal Platform Features (Gains)

Objective: Identify the features and capabilities developers want in a new platform, focusing on what would make their work easier and more efficient.

1. **Desired Platform Outcomes:**
    - "What do you think should be the main goal of a developer platform? Increase speed? Reliability? Alignment? Something else?”
    - **Example outcomes:** Fast feedback loops, easy rollback mechanisms, visibility into application health.

2. **Feature Wishlist:**
    - “If cost/set-up time/training weren’t an issue, are there any tools you wish you had at your disposal?”
    - "If you could build your ideal developer platform, what features would it have?"
    - **Example features:** Pre-configured CI/CD pipelines, automated infrastructure provisioning (e.g., Terraform), continuous alignment checks (e.g., Cortex).

3. **Automation Opportunities:**
    - "What processes would you like to see fully automated?"
    - **Example automations:** Automatic environment provisioning, deployment validation, continuous security scans.

4. **Self-Service Capabilities:**
    - "How important is it for you to have self-service access to resources like environments or databases? What would that look like?"
    - **Example tools:** Pulumi for infrastructure as code, a self-service portal for environment creation.

5. **Monitoring and Troubleshooting Tools:**
    - "What tools or features would help you identify and resolve issues faster?"
    - **Example tools:** Integrated monitoring and logging with Grafana and Prometheus, centralized dashboards for service health metrics, real-time alerts.

6. **Customization and Flexibility:**

   - "How much customization do you need in the platform to tailor it to your team's specific workflows?"
   - **Example preferences:** Flexible CI/CD pipeline configurations, ability to add custom scripts to deployment processes.

Section 5

Validation and Prioritization

Objective: Prioritize the pain points and feature requests to focus on what will deliver the most value to developers.

1. **Feature Prioritization:**

   - "Of all the opportunities we've discussed, which ones would have the most immediate impact on your productivity?"
   - **Example priorities:** Automated deployments, real-time monitoring, pre-configured pipelines.

2. **Top Pain Points to Address:**

   - "Which of the pain points we talked about should we fix first? What would make your day-to-day work easier right away?"
   - **Example issues:** Eliminating manual environment setup, reducing build times, improving deployment visibility.

3. **Biggest Workflow Improvements:**

   - "How would solving these challenges change your workflow? What would be the biggest improvement?"
   - **Example improvements:** Faster time-to-deployment, fewer errors in production, more efficient testing processes.

Section 6

Closing the Interview

Objective: Wrap up the interview and set expectations for next steps.

1. **Final Thoughts:**
    - "Is there anything we haven’t discussed that you think is critical for the platform?"

2. **Thank You & Follow-Up:**
    - "Thank you for your input. Your feedback will directly influence the platform design, and we’ll keep you updated as we progress."

Post-Interview Analysis

After conducting the interviews, the data collected should be analyzed to identify key patterns and insights that will drive the platform design. Here's how to interpret the results and determine what to build:

1. **Identify Critical Developer Jobs**

Look for recurring themes in **developer tasks** that must be supported by the platform.

- **Example:** If multiple developers mention issues with setting up environments, the platform should prioritize self-service provisioning and environment management.

2. **Categorize Pain Points**

Group pain points into categories like CI/CD inefficiencies, infrastructure challenges, or collaboration barriers.

- **Example:** If slow CI/CD pipelines are a major frustration, focus on optimizing build times, possibly by introducing parallel processing or caching techniques in Jenkins.

3. Analyze Desired Features

Pay attention to the features and tools developers consistently request.

- **Example:** If automation is a recurring theme, prioritize features like automated deployments (e.g., with ArgoCD) or infrastructure as code (e.g., Terraform) to reduce manual intervention.

4. Map Gains to Impact

Rank the desired platform features by their potential impact on developer productivity and satisfaction.

- **Example:** If real-time monitoring and troubleshooting were mentioned often, consider integrating tools like Prometheus and Grafana to improve visibility into service health.

5. Prioritize Based on Frequency and Importance

High Frequency, High Impact: Features or issues mentioned frequently and with a high impact should be top priorities.

- **Example:** If multiple teams are frustrated with the manual setup of environments, addressing this with automated environment provisioning tools like **Terraform** or **Pulumi** should be an immediate focus to improve productivity across teams.

cortex

cortex.io