BazelCon 2025 BoF – AI Breaks Everything (Bazel & AI) session summary

Facilitators: Helen Altshuler, Adam Frankl

Summary:
A group of engineers from multiple organizations met to discuss how AI-assisted development is affecting Bazel-based builds. The conversation explored where AI tools work well today (e.g., generating initial BUILD files, refactoring, and large-scale migrations) and where they struggle (understanding Bazel concepts, handling company-specific setups, and dealing with noisy build outputs). Participants reported significant pressure on remote build infrastructure from gigantic AI-related artifacts and dependencies, ongoing challenges around non-deterministic AI agents and context windows, and mixed evidence that AI has improved overall developer productivity. While some early patterns and best practices are emerging (centralized instructions, better prompts, monitoring artifact sizes), attendees agreed that Bazel's strict, deterministic model remains an important stabilizing force amidst rapidly evolving AI-driven workflows.

To participate in future interviews and blogs on the subject - please submit this survey and we will collaborate with you.

Birds of a Feather – AI Breaks Everything

Topics Of Interest

1. AI understanding of Bazel and local context
2. AI-assisted migrations and BUILD file generation
3. Gigantic artifacts and cache pressure
4. Non-determinism, long-running agents, and context windows
5. Toolchain churn, documentation, and AGENTS files
6. Build logs, noise, and AI-assisted debugging
7. Security, stability, and review of AI-generated code
8. Measuring the productivity impact of AI-assisted development

---

1. AI understanding of Bazel and local context

- Strong agreement that “AI doesn’t understand Bazel” out of the box.
- The CLI is too complex for many models: flags and invocation patterns are confusing, and tools frequently guess or hallucinate arguments.
- Teams want AI to lower the barrier to entry for Bazel, but today it struggles with even basic concepts unless carefully prompted (e.g., “What is Bazel?”, “What is a build target?”, “How is a target different from an action?”). AI understand npm really well because there's a lot of open source training data available, but open source Bazel is less common.
- Several participants noted that AI rarely understands *company-specific* Bazel usage without extensive context (org-specific rules, repository layout, conventions, and migration state). What works in one company’s Bazel setup often does not transfer to another.
- One large engineering organization reported that generic chat-based tools run from a browser can answer general questions but cannot understand local context. More focused input from a workspace is often necessary to get useful answers, for example, through an IDE integration.
- Users reported mixed results from building tools that gather context from company chat, email, documents, issue trackers, etc. into AI prompts.
- Markdown “dos and don’ts” files (e.g., “use this test framework, not that one”) plus small example projects as context help tools to help tools behave more consistently, but these are not a complete solution.

---

2. AI-assisted migrations and BUILD file generation

- Multiple teams are using AI to generate or refactor BUILD files, especially during large-scale migrations (e.g., from legacy build systems to Bazel).
- At small scale, AI can write reasonable BUILD files which experienced engineers can sanity-check; at large scale, more conservative approaches are needed.
- In some cases, AI generates the initial configuration for a deterministic tool like Gazelle. In other cases, AI generates and maintains entire BUILD files on its own.
- Example from a large consumer-facing product company:

    - LLMs were used to fix many known migration issues across thousands of repositories and components moving to Bazel, mostly independently with little human intervention.

    - A typical pipeline: run `bazel build` / `bazel test`, inspect logs from orchestration infrastructure, and have AI propose fixes for recurring patterns.

    - AI instructed to log attempts (successful or not) through an MCP server so humans could come back later and analyze.

    - AI sometimes “fixes” problems by disabling tests, so results must be validated.

    - Deterministic tooling and AI are complementary: deterministic rules catch standard cases; AI is used for non-deterministic or previously unseen issues.

- Example from a large infrastructure-focused organization:

    - Given hundreds of Maven `pom.xml` files and full code context via an AI-augmented IDE, AI generated Bazel BUILD files in a short period of time. Generated files had errors and needed manual fixes, but AI saved time.

    - Also tried Gazelle for comparison. Gazelle required more initial setup and also produced some errors requiring manual fixes but was also useful.

    - Follow-up prompts told AI to update deps in BUILD files when adding new dependencies.

- AI has been used to refactor large BUILD files, e.g., splitting a large target into dozens of smaller ones or breaking cyclic dependencies. Results are “pretty good” but still need expert review.

- AI is also used to generate recipes for automated refactoring tools, which can then be applied at scale for code and build migrations.

---

3. Gigantic artifacts and cache pressure

- Several participants raised their hands for the assertion that “gigantic artifacts overwhelm traditional caches,” especially in AI-heavy codebases.

- Main drivers of artifact growth:

    - Large transitive dependency graphs, particularly from ML libraries and toolchains that bundle massive amounts of code.

    - AI-generated code and tests: more code, more tests, more outputs. In one case, an AI tool produced several times more tests than before, putting additional load on the remote build infrastructure and cache.

- Some companies monitor artifact size for every build and publish metrics to dashboards to track growth.

- In certain setups, downloading a huge artifact is slower than rebuilding it locally; this is especially relevant for large AI-related artifacts. In those cases, the cost model for remote cache vs. local execution becomes critical.

- Participants noted that this is not purely an "AI problem": very large SDKs and toolchains exhibit similar behavior, but AI accelerates how quickly such patterns appear.

- AI-assisted dead code deletion can help: AI can detect when a massive library has been imported "just for a constant" and propose smaller replacements, reducing transitive fan-in and artifact size.

- There was interest in having Bazel-level mechanisms (or flags) to avoid fetching overly large artifacts when the rebuild cost is lower, similar to hybrid local/remote strategies that race local execution against remote cache hits.

---

4. Non-determinism, long-running agents, and context windows

- Participants distinguished between non-deterministic *code generation* and deterministic *build pipelines*:

    - It is acceptable for AI code generation to be non-deterministic, as long as Bazel's compile/test pipeline remains deterministic and reproducible ("same source → same bits").

    - The build acts as a strict filter: you can experiment with many generated variants, but only builds that pass are accepted.

- Long-running AI agents tend to degrade over time:

    - When making a repetitive change across a large code base, they work well on the first 10–15 call sites or files, then results become less reliable as context grows.

    - Context accumulation and compression can cause the model to forget specific instructions or earlier decisions.

    - A practical mitigation is to run migration or refactoring tasks one file at a time in a fresh session with a clear, well-structured prompt. This avoids polluted or over-compressed context.

- Some teams are experimenting with models with larger context windows, but those only partially alleviate the problem. Prompt discipline and task chunking remain important.

- AI-generated recipes for rule-based transformations act as a more stable, repeatable mechanism for large-scale changes once a good pattern is identified.

---

5. Toolchain churn, documentation, and AGENTS files

- The assertion that “toolchain churn makes ‘frozen stacks’ a fantasy” did not resonate strongly as a primary concern for most attendees.

- Teams still need to produce source files that Bazel and its rulesets understand, regardless of which AI or editor tools are used upstream.

- A more concrete concern is documentation and knowledge fragmentation for both humans and AI tools:

    - Some organizations have begun introducing files like `AGENTS.md` with instructions for specific AI tools.

    - This can lead to duplication, divergence, and incorrect or outdated knowledge being copied into prompts.

- Emerging best practice:

    - Maintain *one central* instructions file that is authoritative for both humans and AI tools, to avoid conflicting instructions and drift. Don't write different files for different tools.

    - Centralize conventions such as “which test framework to use,” “how to invoke Bazel,” and “how to interpret common errors,” and reference this file from prompts.

---

6. Build logs, noise, and AI-assisted debugging

- Many platform engineers are receiving more poorly researched questions about failed builds:
    - Product engineers paste logs where AI wrote the code or invocation, but neither the engineer nor the AI fully understands why the build failed.
- AI tools frequently mis-handle Bazel invocations:
    - They may hallucinate flags or arguments, miss the remote cache, or bypass Bazel entirely (e.g., calling a compiler directly) when they cannot interpret Bazel's output.
- Bazel's output is widely perceived as too noisy:
    - Users often do not read error messages carefully; they paste logs into chat tools instead.
    - In polyglot builds, each tool formats errors differently, making it harder for AIs and humans to identify the relevant error without additional processing.
- Proposed directions:
    - Build tooling that filters build output, identifies the most relevant lines or error blocks, and presents this as structured input to LLMs.
    - Train specialized models to read Bazel build logs and generate concise, accurate explanations that can then be used by general-purpose LLMs.
    - Publish "recipe books" of useful prompts and examples (e.g., sample BUILD files and container build files) that can be embedded into prompts for Bazel-related debugging and migrations.
- Some experiments show that one model may refuse certain transformations ("I cannot do this"), while another model attempts a transformation but may get details wrong. Recipe books and curated prompts help steer them.

---

7. Security, stability, and review of AI-generated code

- Broad concern that AI-assisted development can amplify insecure and unstable patterns:
    - Models tend to mirror existing style and patterns, so if a codebase already contains insecure patterns, AI will likely propagate them.

    - The problem is similar to having many inexperienced developers contributing simultaneously, but with higher volume.

- Guardrails are critical:
    - Security-sensitive concerns (e.g., permission checks per API endpoint) should be centralized and standardized rather than left to ad hoc implementation in every service.
    - The same guardrails apply to humans and AI: introduce shared libraries, frameworks, and linters rather than relying on local judgment in each file.
- Code review is becoming more time-consuming in the face of large AI-generated diffs:
    - Reviewers do not want to read 1000-line AI-generated changes without strong guarantees or good tooling support.
    - There is increasing interest in better linting, formatting, and possibly mutation testing to ensure tests are meaningful, not just numerous.
- Participants reported that AI can generate large volumes of tests quickly, but the meaningfulness and coverage of those tests are not guaranteed. Mutation testing was mentioned as a potential technique to assess test quality.

---

8. Measuring productivity impact of AI-assisted development

- Direct question: “Has AI actually increased dev productivity?”
- Responses were mixed but leaned skeptical:
    - Some engineers can delegate code they would otherwise write by hand, but this is hard to measure rigorously.
    - PR count is not a reliable productivity metric, and increases in volume may hide declines in quality or maintainability.
- Reports from specific teams:
    - Several organizations reported no “notable” productivity improvement so far; instead, engineers have become better prompt engineers without clear top-line gains.

    - Other participants echoed the difficulty of finding meaningful metrics that capture true productivity, not just activity.

- General sentiment:
    - AI is clearly changing how code and BUILD files are written, but organizations lack robust metrics tying AI usage to better outcomes in build times, reliability, or feature delivery.
    - There is ongoing experimentation, but no consensus yet that AI has definitively improved productivity for Bazel-based teams.

Overall, the session underscored that Bazel's deterministic, reproducible build model remains a crucial anchor as AI accelerates code generation, migrations, and test creation. Teams are actively experimenting with ways to give AI more accurate context, tame gigantic artifacts, and interpret noisy logs, while still relying on Bazel to validate and enforce correctness at scale.