Jenn Strater

- Developer Advocate at Gradle
- Previously a software engineer at various companies
- Berlin <-> Minneapolis, MN, USA

@codeJENNerator
Outline

What is Developer Productivity Engineering?
Current state of Developer Productivity
Achieving Build Happiness
  Measure
  Analyze
  Optimize
  Iterate
What is Developer Productivity Engineering?
The Paradox of Success

Without intervention, as the

- Lines of Code
- No. of developers
- No. of repositories
- No. of dependencies
- No. of support tech stacks

increase, so does the frustration!

@codeJENNerator
How do we get back to bliss?
Creative Flow
Collaborative Effectiveness
Quality of Creative Flow
+ Collaborative Effectiveness

Team Productivity
Developer Productivity Engineering is a culture where the whole organization commits to an effort to maximize developer productivity.
Developer Productivity Engineering KPIs

- Degree of automation
- Speed of feedback cycles
- Correctness of the feedback
Developer Productivity Engineering

Priorities and success criteria primarily based on data that comes from a fully instrumented toolchain.
Current State of Developer Productivity Engineering
Context Switching
The longer the build the harder to debug
In the case of a failure, the time fixing the failure is growing exponentially with the time it takes to detect it.
Because of growing build times, test and builds are pushed to a later point in the life cycle.

The exponential costs for debugging is increased by that.

It also increases the change set size as it becomes inconvenient to get feedback.
Low Developer Productivity is blocking innovation
Breaking the cycle
Step 0. **Focus on Developer Productivity**
Prioritization without data

- Troubleshooting sessions begin with a game of 20 questions
- Person asking for help often doesn’t know what context is important
- Helpers can burn out on helping
- Root cause analysis often impossible without the helper reproducing the problem
- Impossible to determine the impact of the issue
Results in anecdote-driven prioritization

- Non-verification failure masks as verification failure (flakey test)
- Verification failure masks as non-verification failure (snapshot dependency issue)
- Non-verification failure might be caused by bug in a plugin or user mis-configuration
- Many issues are flakey and hard to reproduce

@codeJENNerator
Reliability
Flaky builds and tests are maddening

But I didn’t change anything!

@codeJENNerator
Most developers want to work in an environment that enables them to work at their full potential.

Organizations that cannot provide such an environment will lose talent.
Achieving Build Happiness
Measure!
Faster builds improve the creative flow

<table>
<thead>
<tr>
<th></th>
<th>Team 1</th>
<th>Team 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Devs</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>Build Time</td>
<td>4 mins</td>
<td>1 mins</td>
</tr>
<tr>
<td>No. of local builds</td>
<td>850</td>
<td>1010</td>
</tr>
</tbody>
</table>

- The faster the feedback is, the more often devs ask for feedback
- The more often they ask for feedback, the more fine grained they can work.
### Task Execution

**Path** | **Started after** | **Duration** | **Class** |
--- | --- | --- | --- |
.buildSrc:configuration:compileKotlin UP-TO-DATE | 0.723s | 0.017s | org.jetbrains.kotlin.gradle.tasks.KotlinCompileWithWorkers |
.buildSrc:build:compileJava NO-SOURCE | 0.723s | 0.002s | org.gradle.api.tasks.compile.JavaCompile |
.buildSrc:compileJava NO-SOURCE | 0.723s | 0.002s | org.gradle.api.tasks.compile.JavaCompile |
.buildSrc:kotlinDsl:extractPrecompiledScriptPluginPlugins UP-TO-DATE | 0.723s | 0.002s | org.gradle.kotlin.dsl.provider.plugins.precompiled.tasks.ExtractPrecompiledScriptPluginPlugins |
.buildSrc:binary:CompatibilityOfplugInDescriptors UP-TO-DATE | 0.724s | 0.000s | org.gradle.plugin.devel.tasks.GeneratePluginDescriptors |
.buildSrc:integrationTest:processResources NO-SOURCE | 0.724s | 0.000s | org.gradle.plugin.devel.tasks.GeneratePluginDescriptors |
.buildSrc:performance:processResources NO-SOURCE | 0.724s | 0.000s | org.gradle.plugin.devel.tasks.GeneratePluginDescriptors |
.buildSrc:cleanup:plugInDescriptors UP-TO-DATE | 0.724s | 0.000s | org.gradle.plugin.devel.tasks.GeneratePluginDescriptors |
.buildSrc:packaging:plugInDescriptors UP-TO-DATE | 0.724s | 0.001s | org.gradle.plugin.devel.tasks.GeneratePluginDescriptors |
.buildSrc:binary:CompatibilityOfprocessResources NO-SOURCE | 0.725s | 0.000s | org.gradle.language.jvm.tasks.ProcessResources |
.buildSrc:compileGroovy NO-SOURCE | 0.725s | 0.000s | org.gradle.language.jvm.tasks.ProcessResources |
.buildSrc:build:compileGroovy UP-TO-DATE | 0.725s | 0.014s | org.gradle.api.tasks.compile.GroovyCompile |
.buildSrc:integrationTest:processResources UP-TO-DATE | 0.725s | 0.000s | org.gradle.language.jvm.tasks.ProcessResources |
.buildSrc:processResources NO-SOURCE | 0.725s | 0.000s | org.gradle.language.jvm.tasks.ProcessResources |
.buildSrc:cleanup:processResources UP-TO-DATE | 0.725s | 0.000s | org.gradle.language.jvm.tasks.ProcessResources |
.buildSrc:kotlinDsl:generateScriptPluginAdapters UP-TO-DATE | 0.725s | 0.000s | org.gradle.kotlin.dsl.provider.plugins.precompiled.tasks.GenerateScriptPluginAdapters |
.buildSrc:classes UP-TO-DATE | 0.726s | 0.000s | org.gradle.api.DefaultTask |
.buildSrc:packaging:processResources UP-TO-DATE | 0.726s | 0.001s | org.gradle.language.jvm.tasks.ProcessResources |
.buildSrc:kotlinDsl:plugInDescriptors UP-TO-DATE | 0.726s | 0.000s | org.gradle.plugin.devel.tasks.GeneratePluginDescriptors |
.buildSrc:plugins:extractPrecompiledScriptPluginPlugins UP-TO-DATE | 0.726s | 0.000s | org.gradle.kotlin.dsl.provider.plugins.precompiled.tasks.ExtractPrecompiledScriptPluginPlugins |
.buildSrc:versioning:plugInDescriptors UP-TO-DATE | 0.726s | 0.000s | org.gradle.plugin.devel.tasks.GeneratePluginDescriptors |
.buildSrc:plugins:generateScriptPluginAdapters UP-TO-DATE | 0.726s | 0.000s | org.gradle.kotlin.dsl.provider.plugins.precompiled.tasks.GenerateScriptPluginAdapters |
.buildSrc:versioning:processResources UP-TO-DATE | 0.726s | 0.000s | org.gradle.language.jvm.tasks.ProcessResources |

[@codeJENNerator](https://e.grdev.net/s/vpui4db7vx6xo/timeline)
### Performance Summary

<table>
<thead>
<tr>
<th>Build</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total build time</td>
<td>16.420s</td>
</tr>
<tr>
<td>Initialization &amp; configuration</td>
<td>2.906s</td>
</tr>
<tr>
<td>Startup</td>
<td>0.413s</td>
</tr>
<tr>
<td>Settings</td>
<td>0.066s</td>
</tr>
<tr>
<td>Build :buildSrc</td>
<td>0.486s</td>
</tr>
<tr>
<td>Loading projects</td>
<td>0.007s</td>
</tr>
<tr>
<td>Configuration</td>
<td>1.934s</td>
</tr>
<tr>
<td>Execution</td>
<td>13.514s</td>
</tr>
<tr>
<td>Task execution</td>
<td>10.690s</td>
</tr>
<tr>
<td>End of build</td>
<td>2.824s</td>
</tr>
<tr>
<td>Total garbage collection time</td>
<td>0.609s</td>
</tr>
<tr>
<td>Peak heap memory usage</td>
<td></td>
</tr>
<tr>
<td>G1 Old Gen</td>
<td>0.63/2.62 GB (23.9%)</td>
</tr>
</tbody>
</table>

[https://e.grdev.net/s/vpui4d7vx6xo/performance](https://e.grdev.net/s/vpui4d7vx6xo/performance)
<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>All tasks</td>
<td>1145</td>
</tr>
<tr>
<td>Tasks avoided</td>
<td>447 (62.2%)</td>
</tr>
<tr>
<td>From cache</td>
<td>262 (38.5%)</td>
</tr>
<tr>
<td>Up-to-date</td>
<td>185 (25.8%)</td>
</tr>
<tr>
<td>Tasks executed</td>
<td>270 (37.7%)</td>
</tr>
<tr>
<td>Cacheable</td>
<td>0 (0.0%)</td>
</tr>
<tr>
<td>Not cacheable</td>
<td>270 (37.7%)</td>
</tr>
<tr>
<td>Lifecycle</td>
<td>176</td>
</tr>
<tr>
<td>No source</td>
<td>249</td>
</tr>
<tr>
<td>Skipped</td>
<td>3</td>
</tr>
</tbody>
</table>

Snapshooting task inputs: 966 | 7.726s

1145 tasks in 118 projects:
- distributions:sonZip:3.454s
- :kotlinCompilerEmbeddable:unpackKotlinCompilerEmbeddable:3.118s
- :performance:distributedPerformanceTest:FROM-CACHE:2.292s
- :docs:distDocs:FROM-CACHE:1.153s
- :kotlinCompilerEmbeddable:clean:0.729s
- :toolingApi:toolingApiShadedJar:0.708s
- :javascript:processResources:0.686s
- :kotlinCompilerEmbeddable:patchKotlinCompilerEmbeddable:FROM-CACHE:0.664s
- :processServices:jar:0.589s
- :performance:intTestsImage:0.533s
- :resources:Gcc:classpathManifest:FROM-CACHE:0.482s
- :dependencyManagement:jar:0.350s
- :internalIntegTesting:precompile:FROM-CACHE:0.327s
- :core:jar:0.320s
- :internalIntegTesting:jar:0.257s
- :core:compile:FROM-CACHE:0.201s
- :kotlin:jar:0.199s
- :dependencyManagement:compile:FROM-CACHE:0.162s
- :internalTesting:jar:0.154s
- :modelCore:compile:FROM-CACHE:0.125s
- :dr:jar:0.116s
- :modelCore:jar:0.116s
Capture data from every build run on local AND CI

Comprehensive data allows for root cause analysis without reproducing locally
Analyze!
What is Gradle Enterprise?
Gradle Enterprise is a data platform

- **Collect team data** - all the data about every build across the team creates unique dataset and insights.
- **Build Performance Management** - only with representative, actionable data will builds get and stay fast and reliable.
- **Debugging acceleration** - only with comprehensive, deep data is it possible to quickly discover the root cause for build failures.
Failure Types

- **Verification Failures**
  - Syntax error detected by compilation
  - Code style violation detected by checkstyle
  - Misbehavior in the code detected by a JUnit test

- **Non-Verification Failures**
  - Flaky Test
  - Binary repository down
  - Out of memory exception while running the build

- **Slow Builds**
Impact analysis

Execution failed for task *
> Build cancelled while executing task *

See all failures

Failure count

- 58 out of 74,376 builds

Affected users

- 30 users (51.72%)
- 13 users (22.41%)
- 5 users (8.45%)
- 2 OTHERS (3.45%)

Affected hosts

- 13 hosts (22.41%)
- 11 hosts (18.97%)
- 8 hosts (14.06%)
- 2 OTHERS (3.45%)

Top Tags

- CACHE
- CI/Check.FunctionalTest
- LOCAL
- dirty
- QuickFeedback/LinuxOnly
- johannes/platform-dramonic/language
- jsdelivr.js/partial-to-information-take-2
- QuickFeedback
Root Cause Analysis
Optimize!
Build was run with a development Gradle version (6.0-rc-1)
Development versions of Gradle may be less stable than final versions.
Learn more about the latest Gradle version

Build was run via the daemon

Parallel execution was enabled

Some task outputs were not cacheable
Cacheable task outputs can be reused between builds on one computer or even between builds running on different computers via build cache. Some task outputs were not cacheable due to the following reasons:

Unknown reason
Task output caching not enabled
Learn more about the Task Output Cache

https://e.grdev.net/s/vpui4db7vx6xo/performance/suggestions
Build Caching

When the inputs have not changed, the outputs can be reused from a previous run.

- Gradle Tasks
- Maven Goal Executions
Cacheable Task/Goal Executions

Gradle Compile/Maven Compile
- Source Files
- Compile Classpath
- Java version
- Compiler configuration
- etc...

Gradle Test/Maven Surefire
- Test Source Files
- Runtime Classpath
- Java version
- System properties
- etc...

Caching is a generic feature and applies to all tasks/goals. For IO-bound tasks/goals caching has no benefits (e.g. clean, copy).

@codeJENNerator
Caching is effective for multi-module builds

**pom.xml**

```xml
<modules>
  <module>core</module>
  <module>service</module>
  <module>webapp</module>
  <module>export-api</module>
  <module>security</module>
</modules>
```

**settings.gradle**

```gradle
include 'core'
include 'service'
Include 'webapp'
Include 'export-api'
Include 'security'
```

Builds with a single module will only moderately benefit from the cache.
<table>
<thead>
<tr>
<th>Module</th>
<th>genSource</th>
<th>compile</th>
<th>checkstyle</th>
<th>compile tests</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>service</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>webapp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>export-api</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Red circle: Task/Goal needs to be executed
- Green circle: Task/Goal is retrieved from build cache

@codeJENNerator
Task/Goal needs to be executed

Task/Goal is retrieved from build cache
<table>
<thead>
<tr>
<th></th>
<th>core</th>
<th>service</th>
<th>webapp</th>
<th>security</th>
<th>export-api</th>
</tr>
</thead>
<tbody>
<tr>
<td>genSource</td>
<td>compile</td>
<td>checkstyle</td>
<td>compile tests</td>
<td>test</td>
<td></td>
</tr>
<tr>
<td>compile</td>
<td>test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkstyle</td>
<td></td>
<td>compile tests</td>
<td></td>
<td>test</td>
<td></td>
</tr>
<tr>
<td>compile tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Red: Task/Goal needs to be executed
- Green: Task/Goal is retrieved from build cache

@codeJENNerator
Cache effectiveness

- Even with only a few modules a cache significantly reduces build and test times
- For larger multi-module builds often 50% of modules are leaf modules
  - Build times is reduced by approximately $\frac{1}{n}$ with $n$ being the number of modules
- Checking the inputs and downloading & unpacking items of the cache introduces overhead.
- Overhead is often very small compared to benefits
- Overhead should be measured and monitored too
Gradle CI Builds

Build times are >80% faster for Gradle Core
Dramatically better caching results due to build scans.
Spring Boot build time for compile and unit tests (fully cached)

https://spring.io/projects/spring-boot

<table>
<thead>
<tr>
<th>Duration [min]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20:25</td>
<td>Build Time without Gradle Enterprise</td>
</tr>
<tr>
<td>03:22</td>
<td>Build Time with Gradle Enterprise out of the box (6x, 16.5%)</td>
</tr>
<tr>
<td>02:47</td>
<td>Build Time with Gradle Enterprise after optimizing (7x, 13.6%)</td>
</tr>
<tr>
<td>01:49</td>
<td>Estimated Build Time with Gradle Enterprise with 2019.2 release (11x, 8.9%)</td>
</tr>
</tbody>
</table>

@codeJENNerator
Local Build Cache

- Uses a cache directory on your local machine
- Speeds up development for single developer or build agent
- Reuses build results when switching branches locally

@codeJENNerator
Remote Build Cache

- Shared among different machines
- Speeds up development for the whole team
- Reuses build results among CI agents/jobs and individual developers

@codeJENNerator
Debugging cache misses

LinkedIn: Productivity at scale: How we improved build time with Gradle build cache
SoundCloud: Solving Remote Build Cache Misses by Annoying Your Colleagues

@codeJENNerator
Iterate!
@codeJENNerator
Performance regressions are easily introduced

- Infrastructure changes
- New annotation processors or versions of annotation processors
- Build logic configurations settings
- Code refactoring

@codeJENNerator
What happens today with most regressions

- Unnoticed
- Noticed but unreported
- Reported but not addressed
  - Root cause is hard to detect (especially with flakey issues)
  - Overall impact and priority can not be determined
- Escalated after they have caused a lot of pain
  - Problem gets fixed after it has wasted a lot of time and caused a lot of frustration amongst developers.

**Result:** The average build time is much higher than necessary and continuously increasing.
Performance Analytics
Failures Dashboard
Execution failed for task 
> There were failing tests. See the report at: file://$/path/to/build/reports/index.html

Execution failed for task 
> Too many failures in first run

A problem was found with the configuration of task "type 'IntegrationTest'.
> Directory "target"/"work"/"subprojects"/"build"/"integ-test"/"samples" specified for property 'distribution Samples.samples' does not exist.

A problem was found with the configuration of task "type 'IntegrationTest'.
> Directory 'C:\target\work\a16b87a7070f8c6e\subprojects\"build\"/integ-test\"samples' specified for property 'distribution Samples.samples' does not exist.
Execution failed for task 
> There were failing tests. See the report at: file:///**/build/reports/**/index.html

Failed builds (50 most recent)

Start time: yesterday at 11:48:42 PM
Project: gradle
Requested tasks/goals: clean largeMonolithicJavaProject fullPerformanceTests --baselines 6.1-commit-e17948 --scenarios up...
User: tcagent1
Hostname: dev32.gradle.org
Flaky Tests

Flaky tests are a problem for everyone
We’re currently working on a solution
Plan to release this in Q4/2019
Interested in your ideas & thoughts
Conclusion

- Don’t suffer in silence.
- The Path to Build Happiness is through Developer Productivity.
- Measure, Analyze, Optimize, and Iterate to achieve and maintain build happiness.
Resources

- Early Access Book: https://gradle.com/developer-productivity-engineering
- Try out build scans for Maven and Gradle for free: https://scans.gradle.com
- Gradle Enterprise docs and tutorials: https://docs.gradle.com
Thank you!

@codeJENNerator