AUTHOR

Kwangjin Lee
k.lee-5@student.tudelft.nl

SUPERVISORS

Sebastian Proksch, Shujun Huang

Evolution of CI Pipeline Complexity: Impact on Build Performance

How do CI/CD tool selection, and pipeline complexity affect build performance metrics?

01. Background & Motivation

- As software development increasingly adopts continuous delivery practices, Continuous Integration and Deployment (CI/CD) have become fundamental to modern software engineering. [1]
- Developers face complex decisions when selecting CI/CD tools and managing pipeline architectures. [1]
- Understanding how CI/CD tool selection impacts performance could help organizations make better strategic decisions. [1]

02. Research Question

- **Main Research Question:** How do CI/CD tool selection, and pipeline complexity affect and build performance metrics?
- **Sub-Questions:**
  - How does CI/CD pipeline complexity correlate with build performance metrics in open-source projects?
  - How do significant pipeline evolutionary events, such as CI tool type change, impact performance metrics?
  - Which CI/CD activities contribute most significantly to pipeline complexity in open-source projects?

03. Methodology

- **Data Collection:**
  - 196 open-source projects on GitHub with >100 stars, >5 contributors, last 3 months
  - Java, Python, JavaScript, and other popular languages (Most used languages in 2024) [2]
- **Data Points Collected:**
  - CI Pipeline information from configuration files (only from .github/workflows)
    - GitHub Actions [4]
  - Pipeline Complxity Metrics:
    - Number of jobs, Number of steps
    - Performance Metrics`
    - Build Duration
      - Selected as it impacts develop productivity [1]
    - Build success rate
      - Selected as it ensures code stability [3]
    - Time to Fix Failed Build
      - Time to Fix Failed Build is the duration between a failed run and the next successful run in the same context (i.e., same branch, pull request number, and commit SHA). This metric indicates how quickly developers respond to pipeline failures. [1]
- **Analysis Methods:**
  - Correlation analysis between pipeline complexity and performance metrics
  - Comparison table for migration event comparison

04. Results/Findings

Metric Pair	N	Default Mean±SD	Other Mean±SD	Spearman r	p-value	T-test p
number of jobs vs build duration	194	27.54±219.53	117.18±1062.76	0.37	3.25×10^{-14}	0.8550
number of steps vs build duration	194	27.54±219.53	117.18±1062.76	0.40	1.15×10^{-16}	0.1191
number of jobs vs fix time	70	1070.07±3204.50	2712.33±8873.76	-0.26	0.0822	0.0919
number of steps vs fix time	70	1070.07±3204.50	2712.33±8873.76	-0.16	0.299	0.0907
number of jobs vs success rate	194	0.82±0.24	0.75±0.24	-0.10	0.0587	0.5687
number of steps vs success rate	194	0.82±0.24	0.75±0.24	-0.04	0.466	0.2819

small p-value (typically < 0.05) suggests that the observed correlation is statistically significant and unlikely to have occurred by chance

Repository	Date	Before Migration			After Migration			Change (%)		
		Duration	Fix Time	Success Rate	Duration	Fix Time	Success Rate	Duration	Fix Time	Success Rate
AlexProg_rammerDE/SoulFire	2025-04-01	0.03	15.0	98.9	0.03	416.6	85.0	0.0	2677.33	-14.02
deepspeedai/DeepSpeed	2025-02-24	2.0	1.4	85.2	1.6	5.5	90.0	-20.0	292.86	5.63
hiero-ledger/hiero-sdk-java	2025-01-29	1.8	771.0	63.7	4.4	20.5	72.0	144.44	-97.34	13.05
pandas-dev/pandas	2025-01-21	0.2	0.0	55.3	0.2	151.6	94.1	0.0	∞	70.16
Mean	**Mean**	**1.01**	**196.85**	**75.78**	**1.56**	**148.05**	**85.28**	**31.11**	**996.91**	**18.71**

- No significant improvement
- Project-specific

- 17 repositories average
- ⬇Build Time, ⬇ Fix Time
- ⬆ Build Success Rate (small)

Metric	Before	After	Change Rate (%)
Build Duration	0.68	0.32	-52.9%
Fix Time	202.61	121.41	-40.1%
Build Success Rate	82.18	83.89	2.1%

05. Conclusion

- Increased Pipeline Complexity → Clear Increase in Build Duration
- Weak correlation observed with Build Success Rate & Fix Time.
- Tool change→ Unpredictable & Various Performance Impact
- Benefits are project-specific; no single tool guarantees universal improvement.
- Continuous Maintenance (Config File Changes) → Significant Performance Gains
- ↓ Build Duration (-52.9%)
- ↓ Fix Time (-40.1%)
- ↑ Build Success Rate (+2.1%)
- Key Takeaway: Continuous, strategic optimization is more effective for improving performance than large-scale tool changes.

06. Future work

- Expand Dataset & Scope
- Analyze private and industry projects beyond open-source.
- Investigate other CI/CD platforms besides GitHub Actions.
- Use advanced techniques to better understand the rationale behind pipeline changes.
- Incorporate Qualitative Methods
- Conduct surveys and interviews with development teams to gain deeper context on their design decisions and trade-offs.
- Explore the relationship between pipeline complexity and other software quality attributes beyond build metrics.

References

[1] JetBrains. 2025. Measure CI/CD Performance With DevOps Metrics. In TeamCity CI/CD Guide. Retrieved May 8, 2025, from https://www.jetbrains.com/teamcity/ci-cd-guide/devops-ci-cd-metrics/.
[2] GitHub. 2024. The State of the Octoverse 2024. In GitHub News & Insights. Retrieved May 22, 2025, from https://github.blog/news-insights/octoverse/octoverse-2024/
[3] Software.com. 2024. Build Success Rate. In Engineering Metrics Library. Retrieved May 22, 2025, from https://www.software.com/engineering-metrics/build-success-rate/
[4] JetBrains. 2023. The State of Developer Ecosystem 2023: Team Tools. In JetBrains Developer Ecosystem Survey. Retrieved May 22, 2025, from https://www.jetbrains.com/lp/devecosystem-2023/team-tools/#ci_tools.