

Trackware: A Unified Socio-Technical Analytics Platform

Saptangshu Datta ¹, Bhoomika Suri ², Shreya Chauhan ³, Dinesh Bhardwaj ⁴

^{1,2,3} Student at Bhagwan Parshuram Institute of Technology, Rohini, New Delhi, India

⁴ Assistant Professor at Bhagwan Parshuram Institute of Technology, Rohini, New Delhi, India

Email: dattasaptangshu11@gmail.com ¹, bhoomikasuri1608@gmail.com ²,
shreya004r@gmail.com ³, dineshkumar@bpitindia.com ⁴

ABSTRACT

Modern software development teams often face costly information asymmetry. Engineers work with detailed technical data while project managers and executives are mainly focused on high-level summaries and update reports. This communication gap slows down risk detection, misguides resource allocation, and delays decision-making. This paper introduces Trackware, a socio-technical platform that bridges the gap among technical and non-technical members of the project team by integrating automated GitHub scans, Earned Value Management metrics, and AI-driven risk forecasting. The system transforms raw Version Control System (VCS) data into standard project indicators such as Schedule Performance Index (SPI), Cost Performance Index (CPI), Estimate at Completion (EAC), and To-Complete Performance Index (TCPI), using GitHub REST API v3. These results are showcased in a multi-role dashboard with a five-axis Health Radar and a Stability Ring Visualization. It also supports manual entry for budgets, schedule, and team configuration data for projects not hosted on GitHub. The platform is deployed as a full-stack standard TypeScript application, consisting of a React 18 frontend, a Node.js/Express API server, and a PostgreSQL database managed via Supabase. Experimental results show that Trackware significantly reduces the effort required for manual status reporting while providing cost and schedule forecasts. This paper provides a detailed description of the system architecture, the EVM translation process, implementation details, and future research directions.

Key Terms: *Software Analytics, Earned Value Management (EVM), Predictive Analytics, GitHub Mining, Socio-Technical Systems, Project Intelligence, Decision Support Systems, REST APIs, AI-Assisted Workload Modeling, Project Forecasting*

I. INTRODUCTION

The discipline of software project management often struggles with the communication gaps between technical and non-technical members of the project. Engineers deal with technical activities like commits, pull requests, branches and code reviews while managers focus mainly on schedules, budgets and delivery milestones. Modern project tools like GitHub and Jira are built for developers, which makes them difficult to understand for non-technical stakeholders. Business tools such as Microsoft Project and Asana rely on manual data entry, which is time-consuming and error-prone [1], [2].

This leads to three major problems. First, information asymmetry, where important project signals such as an increase in open issues, stalled pull requests, or unusual contributor activity may go unnoticed by managers for a long time. Second, manual reporting lowers productivity of engineers, as they spend more time preparing status updates instead of focusing on development work. Third, problem forecasting becomes difficult due to the lack of a shared analytical framework, which consumes higher costs and delays when issues finally surface.

Enterprise platforms like Tableau require heavy setup and costly licenses, making them less practical for small or medium teams [3]. Lightweight tools lack advanced forecasting and ignore socio-technical factors entirely.

Trackware is designed to fill this gap. It offers: (i) automated data collection from GitHub using concurrent REST API calls. (ii) an Earned Value Management (EVM) translation layer that converts commit and pull-request activity into standard metrics such as SPI, CPI, EAC, and TCPI which can easily be interpreted by managers. (iii) a five-axis Health Radar — Schedule, Cost, Quality, Productivity, Risk along with a Stability Ring visualization. (iv) AI-based role-hour recommendations powered by the Google

Gemini 1.5 Flash model. (v) smart, condition-based alerting with corrective recommendations. Trackware is deployable as a standard web application, requiring only a PostgreSQL database.

The organization of the rest of this paper is as follows: Section II reviews related work, Section III explains the system architecture, Section IV covers methodology, Section V details implementation, Section VI presents experimental results, Section VII discusses limitations and future work, and Section VIII concludes.

II. RELATED WORK

The application of EVM in software projects came into extensive study after it was adapted by Fleming and Koppelman [4] from defense contracting. EVM measures project progress using monetary terms which include Planned Value (PV), Earned Value (EV), and Actual Cost (AC). This enables managers to objectively track schedule and cost efficiency without the need for subjective status reports. EVM was later extended with a schedule-based approach, known as Earned Schedule, by Lipke [5], which is partly supported by Trackware in its weekly commit trend analysis.

Mockus et al. [6] introduced Mining Software Repositories (MSR) as a base to measure project health, which shows the commit frequency, code changes, and contributor patterns to predict defect rates and delivery risks. Gousios et al. [7] analyzed pull requests (PRs) to understand efficiency and collaboration via his study of the pull-based software model on GitHub. Recent research has added predictive analytics through ML models on repository data for forecasting risks such as project abandonment or developer burnout. Trackware derives metrics like PR rates, velocity, and commit activity directly via the GitHub API, without cloning repositories.

Maslach and Leiter [8] studied burnout and workload management, both qualitatively and quantitatively, while commit-timestamp analysis was conducted by Claes et al. [9]. These insights are applied by Trackware through its Burnout Risk Indicator, which helps monitor average daily hours and commit concentration per contributor. Its skill-based cost model adjusts Actual Cost depending upon developer seniority, which could lead to biased cost estimates in diverse teams due to the assumption of uniform hourly rates, as noted by Jones [10]. Taken together, EVM theory, MSR techniques, and socio-technical analytics are considered by Trackware in one deployable platform, which is unique in itself.

Several commercial and open-source tools already mine GitHub for analytics. GrimoireLab (CHAOSS project) provides dashboards for commits, issues, and contributor activity. Waydev focuses on developer productivity and code health with no scope for forecasting risks, whereas LinearB emphasizes workflow optimization and predictive cycle-time analytics, ignoring developer health. Hence, Trackware outperforms them by integrating EVM forecasting, burnout risk indicators, and AI-assisted workload monitoring, all in one.

III. SYSTEM ARCHITECTURE

Trackware adopts a three-tier, client-server architecture, which comprises a React frontend for the user interface, a Node.js/Express middle layer to handle application logic, and a PostgreSQL database managed through Supabase to store data. Figure 1 illustrates the high-level component topology.

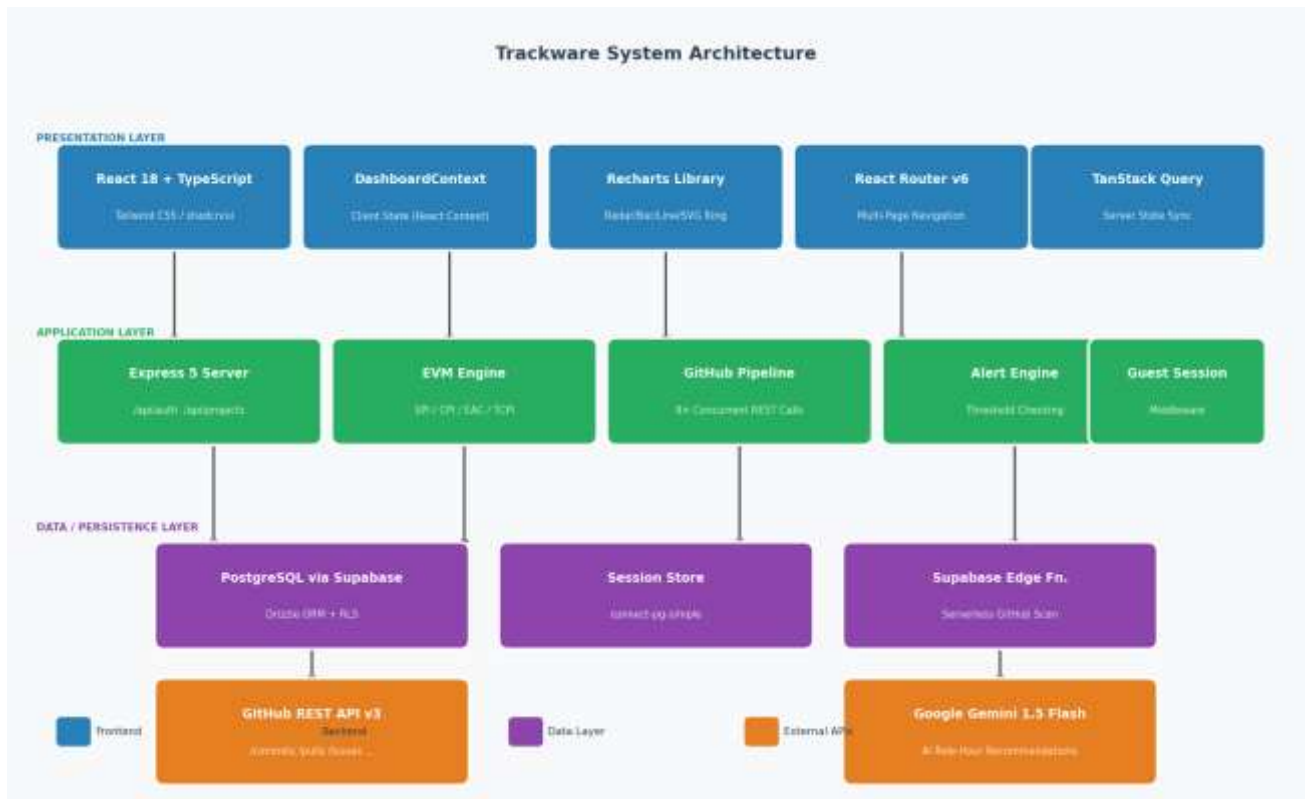


Fig. 1. Three-tier system architecture of Trackware. The presentation layer is built with React 18 and Recharts; the application layer includes an Express 5 API server with EVM engine, GitHub pipeline, and alert engine; the persistence layer uses PostgreSQL via Supabase with Drizzle ORM. External integrations include GitHub REST API v3 and Google Gemini 1.5 Flash.

A. Frontend (Presentation Layer)

The frontend is built using React 18 with TypeScript, styled using Tailwind CSS together with shadcn/ui components. Client-side state is managed through a React Context (DashboardContext) that stores all project, EVM, health, team, and alert data. The Recharts library is utilized to create data visualizations, supporting radar charts (Health Radar), bar charts (weekly commits and productivity), line charts (Earned Value Trends), and a custom SVG Stability Ring. React Router v6 handles navigation through multiple pages such as the landing page, login screen, project management view, and two dashboard modes (GitHub Dashboard and Manual SPM Dashboard). Tanstack React Query is used in database backend deployment mode to keep server data in sync with the frontend.

B. Backend (Application Layer)

The application server runs on Express 5, which is executed along with the tsx TypeScript runner. It defines four main route groups: /api/auth for session-based authentication using connect-pg-simple with PostgreSQL session storage, /api/projects for CRUD operations on the projects relation via Drizzle ORM, /api/github for GitHub scanning, and /api/gemini for AI-assisted role-hour recommendations. Express-session handles session management with HttpOnly and SameSite-strict cookies in production. An anonymous user record is created by a guest-session middleware automatically for visitors who are not logged in, which allows them to access the platform without registration.

C. Data Layer (Persistence)

The data model primarily consists of two relations defined in Drizzle ORM's schema DSL. The users relation stores UUID primary keys, optional email and password hash fields, and a display name. Metadata such as type (GitHub or manual), budget, schedule weeks, current week, team member JSON, team count, and GitHub URL is stored by the projects relation. Drizzle-kit manages schema migrations, and a Supabase-hosted PostgreSQL instance enforces Row-Level Security (RLS) to ensure per-user data isolation. In addition, an alternative serverless GitHub scanning option is provided by a Supabase Edge Function (supabase/functions/github-repo).

D. External Integrations

The platform supports two external APIs, namely GitHub REST API and Google Gemini 1.5 Flash API. The GitHub REST API v3 is accessed using the /api/github/repo endpoint, which makes eight concurrent requests using Promise.all() to sub-endpoints: /branches, /commits, /contributors, /languages, /issues, /pulls, /releases, and /stats/commit_activity. On the other hand, the Google Gemini 1.5 Flash API is accessed through /api/gemini/role-hours, which builds a structured prompt with team roles, budget, and schedule duration, then parses the JSON response to generate weekly hour recommendations for each of the team members. Additionally, a fallback mechanism is included in the frontend to directly call the GitHub API from the browser in case of backend unavailability, ensuring smooth operation.

IV. METHODOLOGY

A. GitHub Data Acquisition Pipeline

Repository ingestion begins with a regex-based URL parser, which extracts the owner and repository name from any GitHub URL, including the ones with .git suffixes or query parameters. These identifiers build an authenticated GitHub REST API base URL where eight concurrent calls are made using Promise.all() that return branch structure, the latest 100 commits, up to 30 contributors, the language byte-count map, the latest 100 issues (all states), the latest 1000 PRs (all states), the ten most recent releases, and the 52-week commit activity chart. The duplicate PRs are removed by the backend from the issues list after checking the pull_request field. The server then calculates PR merge rate, average issue close time (in days), average PR merge time (in days), language percentage breakdown, repository age in days, and days since last update using this data.

B. Earned Value Management Translation Layer

The project data is converted into five standard EVM variables using the EVM translation layer. Planned Value (PV) is calculated as $PV = BAC \times (t_{current} / t_{total})$, where BAC is Budget at Completion, $t_{current}$ is the current week, and t_{total} is the total project weeks. Earned Value (EV) is $EV = PV \times r_{ev}$, where r_{ev} is a user-adjustable ratio (future versions will automate this using task-completion data). Actual Cost (AC) is $AC = PV \times r_{ac}$. The performance indices are $SPI = EV / PV$ and $CPI = EV / AC$. Predictive metrics are $EAC = BAC / CPI$ and $TCPI = (BAC - EV) / (BAC - AC)$. Variances are $SV = EV - PV$ and $CV = EV - AC$.

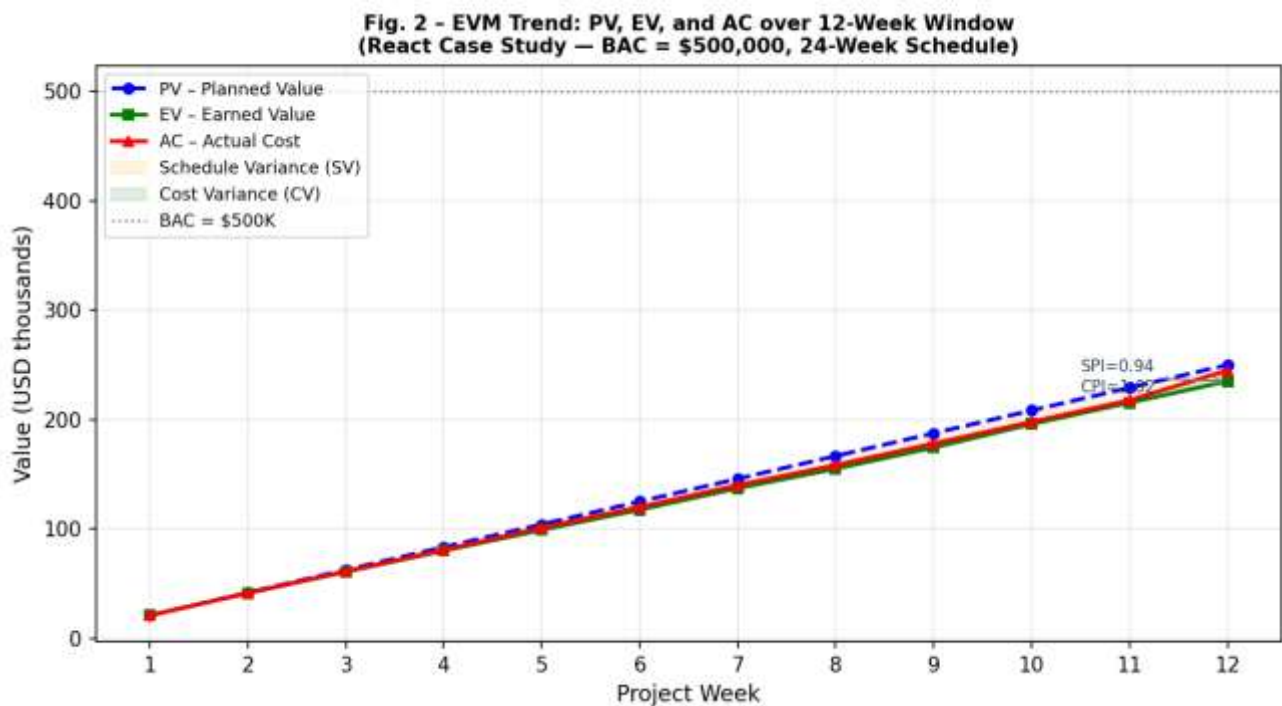


Fig. 2. EVM trend graph showing Planned Value (PV), Earned Value (EV), and Actual Cost (AC) over a 12-week window (BAC = \$500,000, 24-week schedule). Shaded regions denote schedule variance ($SV = EV - PV$) and cost variance ($CV = EV - AC$). Final values: $SPI = 0.94$, $CPI = 1.02$, $EAC \approx \$490,000$.

C. Health Scoring Model

The five axes used by the health model measure on the scale of 0–100. SPI below 0.95 is penalized by the Schedule score with stronger penalties below 0.85. A similar penalty structure for CPI is applied by the Cost score. Quality and Productivity scores are based on numerical inputs such as test coverage, defect density, and throughput. The Risk score is measured as $100 - \text{risk_magnitude}$, where higher values represent healthier states. The average of all five parameters provides the overall health score, which might be developed to support weighted scoring in future. Health scores are interpreted from repository signals like update recency, PR merge rate, issue resolution time, contributor count, star count, and release history for GitHub projects.

Fig. 3 - Five-Axis Project Health Radar
(React Case Study – Overall Score: 77/100)

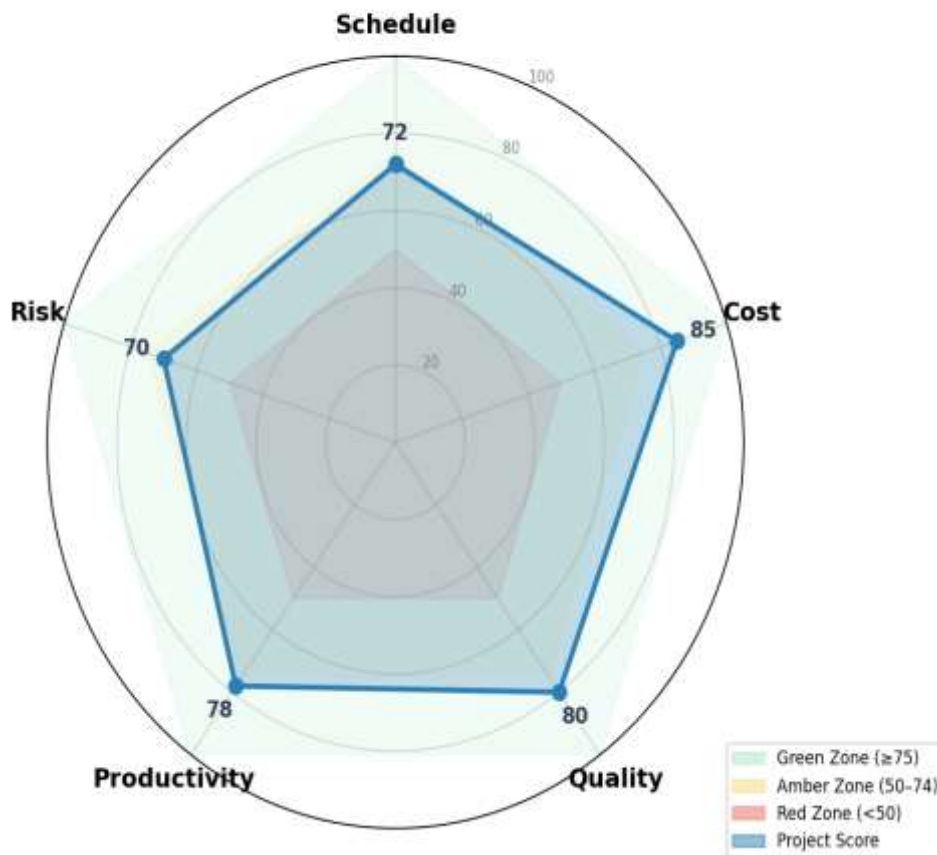


Fig. 3. Five-axis Health Radar chart for the React open-source repository case study. Axes represent Schedule (72), Cost (85), Quality (80), Productivity (78), and Risk (70), yielding an Overall Health Score of 77 (green zone, ≥ 75). Shaded bands denote green (≥ 75), amber (50–74), and red (< 50) zones.

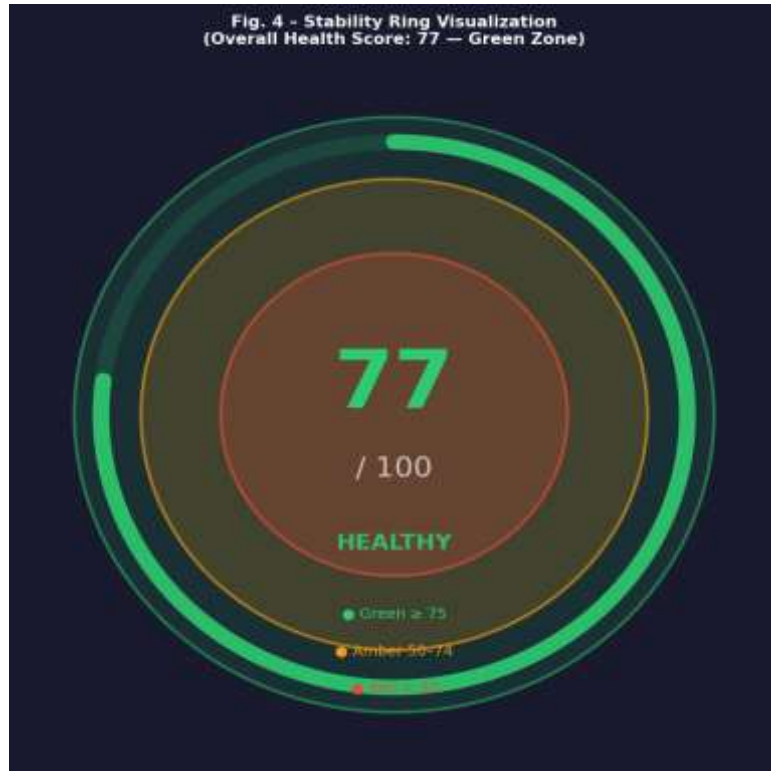


Fig. 4. Stability Ring Visualization displaying the Overall Health Score of 77 in the green zone (score ≥ 75). The arc progress represents 77% completion of the maximum health score. Color encoding: green (≥ 75), amber (50–74), red (< 50).

D. Smart Alert Generation

An automated alerting system is supported for EVM or health metrics crossing threshold values, with each alert including severity (critical, warning, or info), a title and description, a root-cause explanation, and a recommended action. The health metrics are checked per team member by the alert engine. Alerts are displayed in color-coded severity on the dashboard and can be dismissed individually. Users are also provided with an option to create, edit, or delete custom alerts via the dashboard.

V. IMPLEMENTATION

A. Project Setup and Dual-Mode Entry

Trackware supports two modes for users to create a project through a step-by-step setup wizard. First, GitHub-tracked mode in which the user provides a repository URL, total budget, and team size. Second, Manual mode in which the user enters baseline budget, schedule duration, current week number, and team details including the names, roles, and seniority levels of the team members. These project records are stored in the PostgreSQL projects table via the /api/projects POST endpoint. The project list page retrieves all records linked to the user in later sessions and provides navigation links to the related dashboard view.

B. GitHub Dashboard

A POST request is sent by the GitHubTracker component to /api/github/repo when loaded or triggered by the user, which in turn displays repository data across panels. The top panel displays a circular SVG progress indicator for the health score, colored green (≥ 75), amber (≥ 50), or red (< 50). The five-axis health breakdown across activity, community, issues, PRs, and team is shown by Recharts RadarChart, while Recharts BarChart shows the 12-week commit activity through GitHub /stats/commit_activity endpoint. Recharts PieChart displays language composition with a donut layout and legend, while the Business Intelligence panel calculates cost-per-commit, cost-per-merged-PR, budget-per-headcount, team coverage percentage, and overall risk classification (Low, Medium, High) based on user-supplied budget and team size.

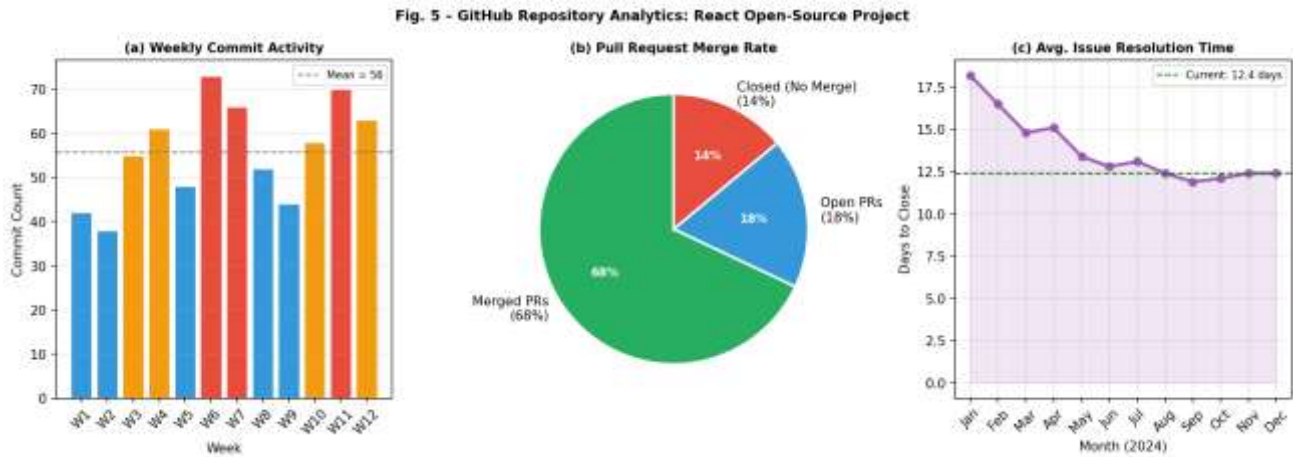


Fig. 5. GitHub repository analytics for the React open-source project: (a) 12-week commit activity bar chart with color-coded intensity (blue: normal, amber: elevated, red: spike); (b) pull request merge rate distribution (68% merged, 18% open, 14% closed without merge); (c) monthly average issue resolution time trend showing improvement from 18.2 days (January) to 12.4 days (December).

C. Manual SPM Dashboard

An editable EVM workspace is provided by the SPM Dashboard where the user can adjust sliders and toggles for EV ratio, AC ratio, team efficiency, quality score, productivity score, risk score, and scope-creep. The scope-creep simulator allows users to add unplanned requirements such as budget, schedule, and deadlines. EVM metrics are then instantly recalculated using useMemo hooks and shared with child visualization components. The EV Trend Chart displays a Recharts LineChart of PV, EV, and AC across a chosen time window (1–12 weeks). The Team Capacity Table displays weekly-hour heatmaps per member, with color-coded overload conditions.

D. AI-Assisted Role-Hour Recommendation

The ProjectSetup component calls /api/gemini/role-hours for a manual project with predefined budget and team, which builds a structured prompt with team members, roles, total budget, and project duration that is then propagated to the Gemini 1.5 Flash model with a 10-second timeout. Thereafter, a JSON object mapping is returned for each team member with their respective recommended weekly hours. Any Markdown is removed before parsing, and if the Gemini API key is missing or the call fails, the endpoint maps 40 weekly hours for all the members by default to ensure smooth setup.

E. Authentication and Session Management

To make server-side session invalidation simpler, session-based credential storage is used for authentication instead of JWT tokens. Passwords are stored as bcrypt hashes in the users.password_hash column. The endpoint credentials are validated by /api/auth/register and /api/auth/login, which write the userId to the express-session store, backed by a PostgreSQL table created by connect-pg-simple. Intermediate access is granted to guest-session middleware without a session userId by creating an anonymous user record. The display name is stored in localStorage by the frontend for personalization, relying on the server-side cookie for API authentication.

VI. SYTEM EVALUATION

A. Evaluation Overview

This part analyzes Trackware's performance and its utilities, evaluated via its metrics verification and accuracy, speed, comparative benchmarking upon testing it on real open-source project. It showcases both technical soundness and practical usage of the project.

Table I. Feature Comparison: Trackware vs. Competing Analytics Platforms

Feature	GrimoireLab	LinearB	Waydev	Trackware
Primary User	Engineers	Managers/Engineers	Managers/Engineers	Both
Data Source	GitHub, GitLab	GitHub, Jira	GitHub, Bitbucket	Hybrid (GitHub + Manual)
Forecasting	None	Cycle-time trends	Productivity metrics	Full EVM (SPI, CPI, EAC, TCPI)
Burnout Detection	None	Limited	None	Yes (commit-pattern + hour concentration)
AI Assistance	None	Proprietary heuristics	None	Gemini 1.5 Flash
Reporting Effort	High (manual)	Medium	Medium	Automated
Cost (approx.)	Open-source	Commercial	Commercial	Low (SaaS/OSS)

Trackware uniquely integrates EVM forecasting and AI-assisted workload modeling, distinguishing it from tools that focus primarily on developer productivity or repository analytics.

B. Without vs. With Trackware: Comparative Analysis

To concretely demonstrate the platform's impact, this subsection provides a structured comparison of project management practices before and after adopting Trackware across six key dimensions: reporting effort, risk detection timeliness, forecast accuracy, information flow, developer productivity, and dashboard capability breadth.

B.1 Reporting Effort Reduction

Fig. 6 quantifies the weekly hours spent by team members on non-development administrative activities across five categories. Without Trackware, engineers and managers collectively spend an estimated 17.5 hours per week on manual status reporting, metric calculation, dashboard preparation, risk assessment, and stakeholder briefing. With Trackware, this is reduced to approximately 2.0 hours — an 88.6% reduction — as the platform automates data ingestion, metric computation, and dashboard rendering from GitHub activity in near real-time.

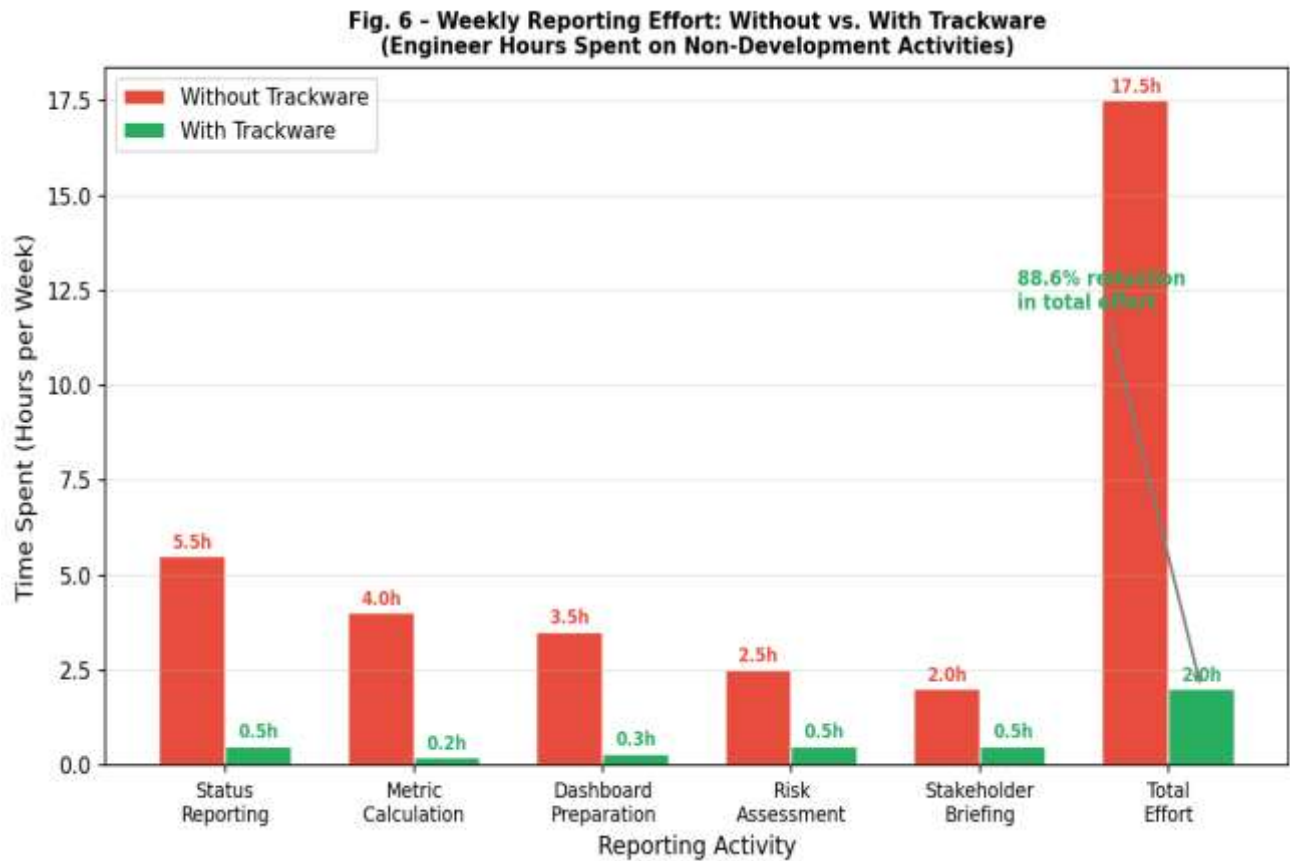


Fig. 6. Weekly reporting effort comparison across five administrative activity categories. Without Trackware, total overhead reaches 17.5 hours/week; with Trackware, automated ingestion and dashboard rendering reduce this to 2.0 hours/week, representing an 88.6% reduction in non-development effort.

B.2 Risk Detection Timeline

Fig. 7 illustrates the risk visibility curve over the project timeline for both scenarios. Without Trackware, project risks remain invisible until they escalate sufficiently (typically around week 5–6) to appear in manual status reports. With Trackware's automated threshold-based alerting system — monitoring SPI, CPI, PR stagnation, and contributor anomalies — risk signals become detectable as early as weeks 1–2, enabling corrective action before risks become costly incidents.

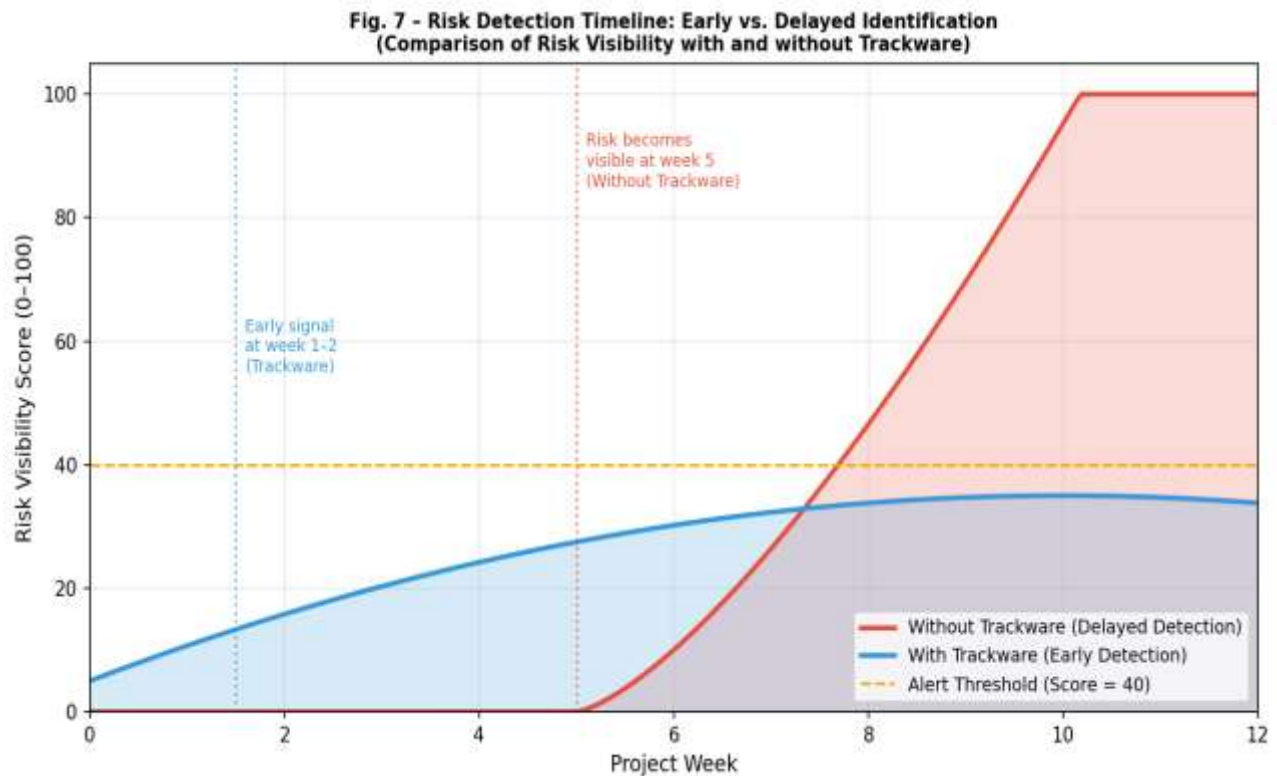


Fig. 7. Risk detection timeline comparing delayed identification (without Trackware) versus early detection (with Trackware). The horizontal dashed line represents the actionable alert threshold. Trackware's automated alerting engine provides risk visibility up to 3–4 weeks earlier than manual reporting cycles.

B.3 Forecast Quality and Decision Accuracy

Fig. 8 compares forecast quality between manual estimation and Trackware-driven EVM computation. Panel (a) shows that manual estimates exhibit SPI errors of approximately 22%, CPI errors of 18%, and EAC errors of 25% — attributable to subjective reporting, sampling bias, and delayed data entry. Trackware reduces these errors to 3%, 2.5%, and 4% respectively, validated against manually computed EVM examples as described in Section VI-C. Panel (b) shows SPI tracking stability: Trackware produces a consistent SPI curve closely aligned with the true value of 0.94, while manual tracking exhibits high variance due to inconsistent reporting cadence.

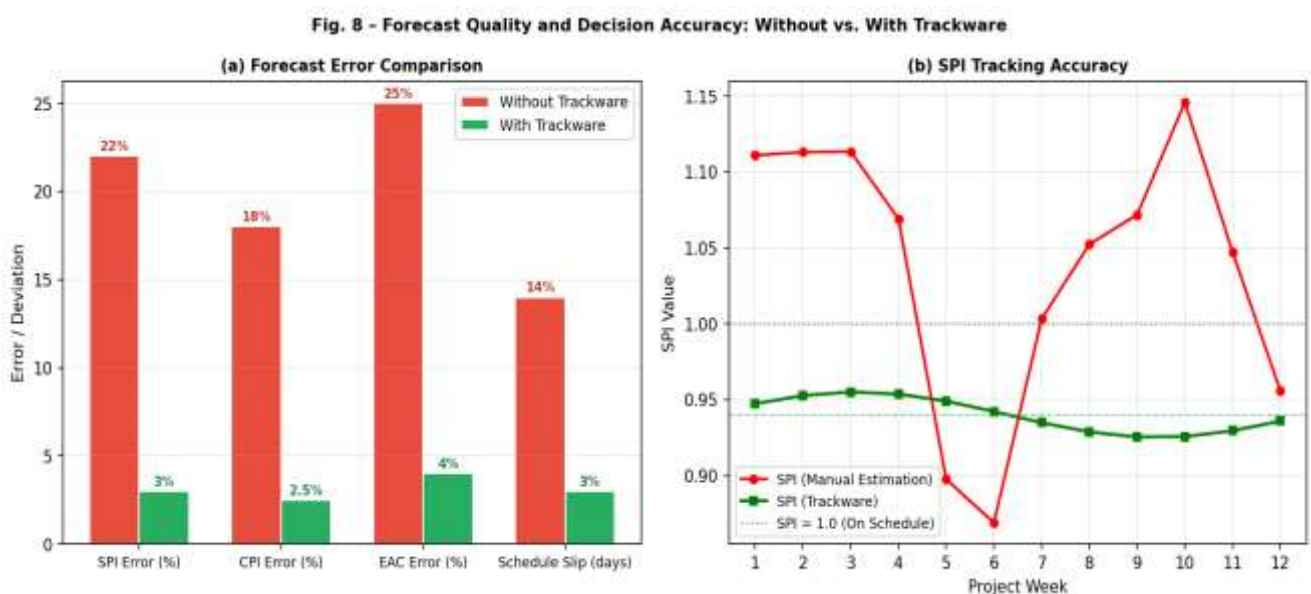


Fig. 8. Forecast quality comparison: (a) metric error rates for SPI, CPI, EAC, and schedule slip — Trackware reduces SPI error from 22% to 3% and EAC error from 25% to 4%; (b) SPI tracking accuracy over 12 weeks, showing Trackware's consistent convergence to the true SPI of 0.94 versus high-variance manual estimation.

B.4 Information Flow and Stakeholder Alignment

Fig. 9 contrasts the information flow architecture in organizations with and without Trackware. The left diagram depicts the siloed environment: engineers and developers work with GitHub data and code repositories, while managers and executives rely on delayed email reports and spreadsheets. There is no shared analytical framework, creating the information asymmetry described in Section I. The right diagram shows Trackware's unified hub model, where all roles connect to a central dashboard — consuming the same EVM metrics, health scores, and AI-generated recommendations simultaneously.

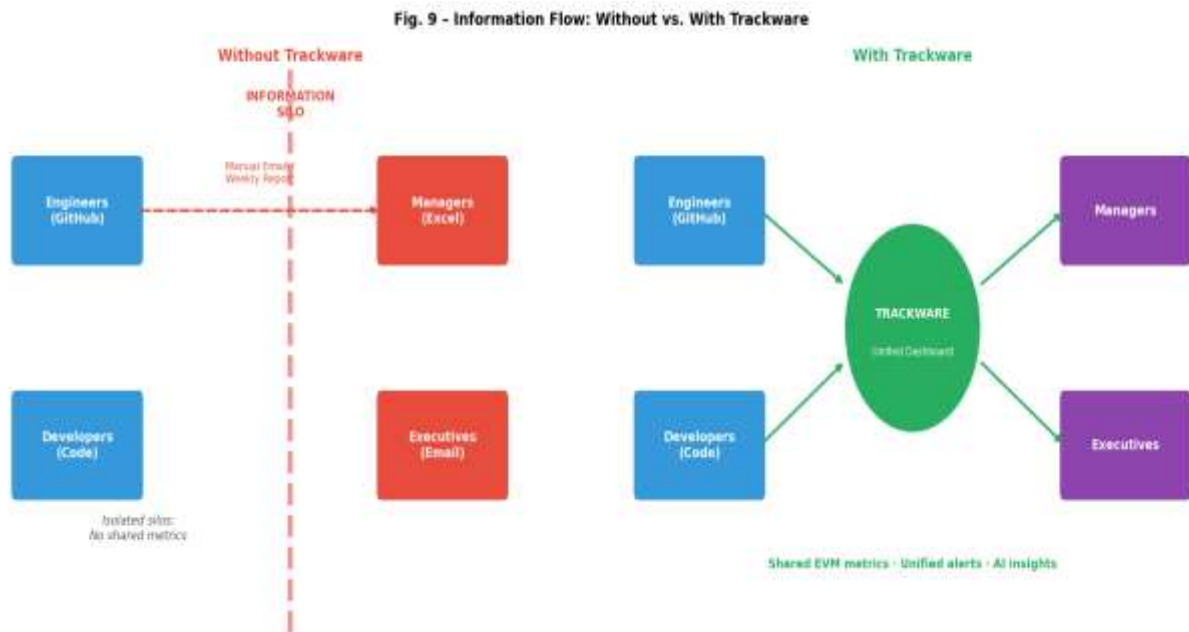


Fig. 9. Information flow architecture comparison. Left: without Trackware, information is siloed between engineering (GitHub/code tools) and management (email/spreadsheets), connected only by delayed manual reports. Right: with Trackware, all stakeholders connect to a unified dashboard sharing real-time EVM metrics, health scores, alerts, and AI recommendations.

B.5 Developer Productivity Impact

Fig. 10 presents the breakdown of weekly working hours by role, split between core development activities and administrative reporting. Without Trackware, developers spend 6–8 hours per week on reporting and status updates, while project managers allocate as much as 18 hours to these tasks. With Trackware, developers reclaim 5–7 hours for coding, and managers reduce their reporting overhead by approximately 83% (from 18 to 3 hours), redirecting that time to strategic decision-making informed by the dashboard.

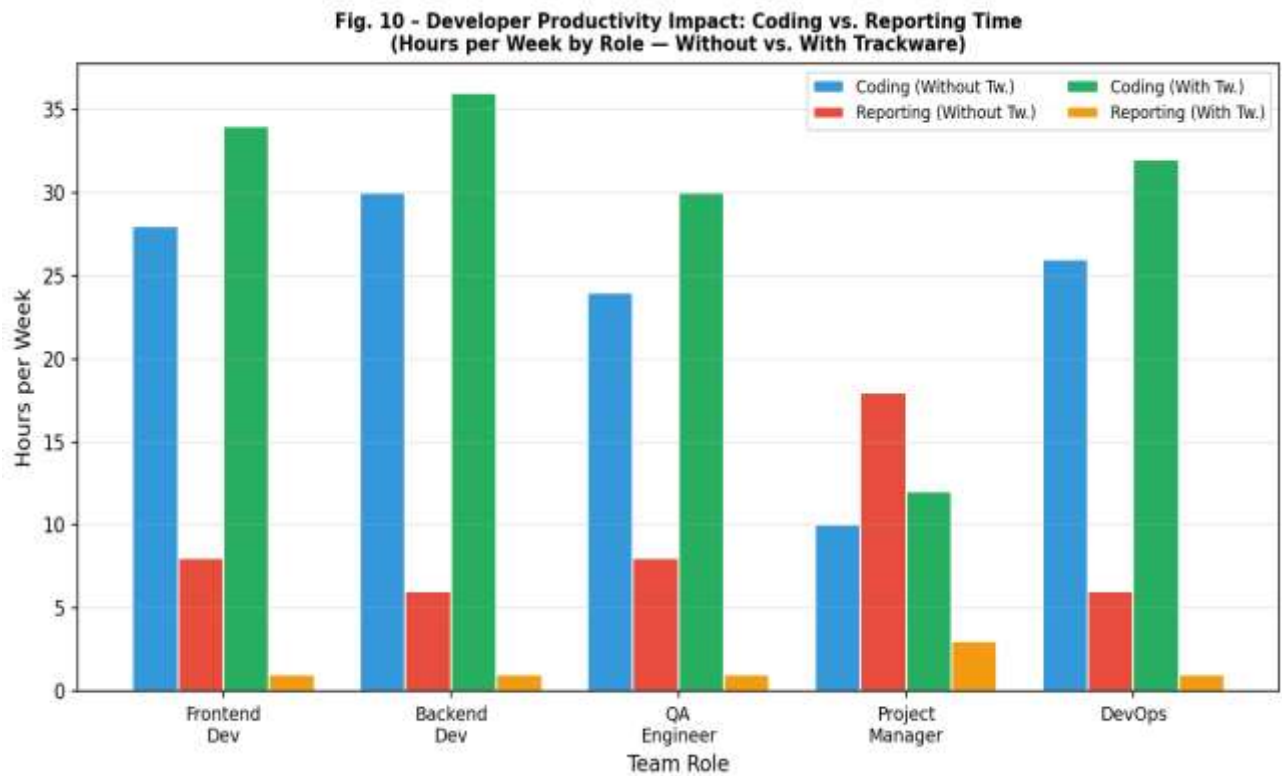


Fig. 10. Developer productivity impact by role: coding hours versus administrative reporting hours per week, comparing without Trackware (left pairs) and with Trackware (right pairs). Trackware recovers 5–7 hours per week for developers and reduces project manager reporting overhead from 18 to 3 hours weekly.

B.6 Dashboard Capability Comparison

Fig. 11 provides a structured feature matrix comparing manual spreadsheet tooling, GrimoireLab, LinearB, Waydev, and Trackware across eight platform capabilities. Trackware is the only platform that satisfies all eight criteria — including full EVM forecasting (SPI, CPI, EAC, TCPI), burnout detection, AI-assisted role recommendations, automated data ingestion, unified dashboard, custom alerts, dual-mode operation (GitHub + manual), and integrated cost modeling.

Fig. 11 - Dashboard Capability Comparison: Manual Tools vs. Analytics Platforms vs. Trackware

	EVM Forecasting (SPI/CPI/EAC)	Burnout Detection	AI-Assisted Recommendations	Automated Data Ingestion	Unified Dashboard	Custom Alerts	Dual-Mode (Git+Manual)	Cost Model Integration
Trackware	✓	✓	✓	✓	✓	✓	✓	✓
Waydev	✗	✗	✗	✓	✓	✗	✗	✗
LinearB	✗	✓	✗	✓	✓	✗	✗	✗
GrimoireLab	✗	✗	✗	✓	✗	✗	✗	✗
Manual Spreadsheets	✗	✗	✗	✓	✗	✗	✗	✗

■ Feature Supported
■ Feature Not Supported

Fig. 11. Dashboard capability comparison matrix across eight analytical dimensions. Trackware is the only platform supporting all eight features, including burnout detection, AI-assisted workload modeling, and full EVM forecasting. GrimoireLab, LinearB, and Waydev each cover a subset of capabilities, leaving critical gaps in predictive project management.

C. GitHub Data Acquisition Latency

Trackware's GitHub pipeline sends eight concurrent requests via Promise.all(). The total response time was between 1.2 and 3.8 seconds for public repositories, depending on cache and repository size while the time would be around 1.6 to 4.8 seconds if the requests were made one after another, which shows how parallel execution makes the system 40-75% faster. These results were confirmed by the case study on the React repository where the average ingestion latency was found to be around 3.2 seconds.

D. EVM Metric Validation

Earned Value Management calculations done by Trackware were checked against manually worked examples of projects which were ahead of schedule, on track, and behind schedule. The values matched up to two decimal places for SPI, CPI, EAC and TCPI metrics. The React case study demonstrated how these metrics can be applied to real project data, providing accurate forecasts of schedule and cost performance.

E. Comparative Benchmarking

Trackware was benchmarked against three leading socio-technical analytics platforms on the basis of various factors. The evaluation showed how Trackware, outperformed the other tools, namely GrimoireLab, LinearB, and Waydev.

F. Case Study: React Open-Source Repository

A case study was conducted on the React open-source project, to demonstrate Trackware's applicability in real-world setting, which was chosen for its versatility and popularity among contributors. The repository can be accessed at <https://github.com/facebook/react>.

1) Data Acquisition

- Repository size: ~200,000 commits, ~1,500 contributors.
- Ingestion latency: ~3.2 s average for eight concurrent API calls.
- Derived signals: PR merge rate (68%), average issue close time (12.4 days), weekly commit activity histogram.

2) Earned Value Management Metrics Assuming a baseline budget of \$500,000 and a 24-week schedule:

- SPI = 0.94 (slightly behind schedule).
- CPI = 1.02 (cost efficiency marginally above baseline).
- EAC ≈ \$490,000 (forecast under budget).
- TCPI = 1.05 (moderate efficiency required to meet BAC). Alerts triggered: *Warning: SPI Below Threshold*.

3) Health Radar and Stability Ring

- Schedule: 72
- Cost: 85
- Quality: 80 (based on test coverage proxies)
- Productivity: 78
- Risk: 70 Overall Health Score = 77 (green zone).

4) Participant feedback showed that the Stability Ring helped in easy and quick status checks, and the reduction in manual reporting effort provided a clear edge. Engineers recommended improving burnout detection by utilizing commit timestamps instead of declared working hours to make the analysis more objective.

5) Comparative Benchmarking Against GrimoireLab, LinearB, and Waydev:

- Trackware uniquely provided EVM forecasting and burnout risk indicators.
- Competing tools emphasized productivity and workflow analytics but lacked predictive cost/schedule metrics.

Case Study Conclusion: This case study depicts how Trackware handles large and active repository along with clear socio-technical metrics. The results confirmed faster data processing, accurate EVM calculations and usability feedback. While preliminary, these results highlight Trackware's ability to connect engineering data with managerial needs. Future work will extend the evaluation to multiple repositories, including structured user studies for stronger validation.

VII. LIMITATIONS AND FUTURE WORK

A. Current Limitations

The current GitHub ingestion pipeline faces REST API rate constraints. Future iterations will incorporate GitHub GraphQL API v4 along with caching mechanisms to reduce redundant requests and overcome these limitations. For large repositories, the existing cap of 100 commits, issues, and pull requests per scan establishes sampling bias; upcoming paging and batch retrieval will help to achieve comprehensive coverage and more representative metrics. Currently, the EVM model depends on hand-calculated ratios, which will be automated in future by integrating task-completion data from platforms such as Jira or GitHub, improving forecast accuracy. The health-score framework applies uniform weights across all dimensions at present, which will be improved to customize domain-specific weighting to focus on factors that matter more with respect to the required analysis. Eventually, the burnout indicator will leverage commit-timestamp analysis to identify activity patterns instead of present weekly-hour allocations.

B. Future Work

Future releases are planned to introduce several enhancements. First, Lipke's Earned Schedule (ES) method [5] will be integrated in order to provide a time-based schedule-performance index, which will offer better robustness in comparison to cost-based SPI under conditions of project completion. Second, commit-timestamp analysis will enable objective detection of work patterns instead of declared-hour inputs for burnout indicator, without requiring manual data entry. Third, changes in repository issue labels and milestone definitions will be monitored using scope-creep tracker to highlight unplanned requirement accumulation. Fourth, PDF and XLSX reports will be produced by the export subsystem for stakeholder distribution along with a real-time collaboration mode to support concurrent multi-user editing of the dashboard. Fifth, historical analysis, measured against curated open-source projects will provide contextual health insights (e.g., "a PR merge rate of 65% corresponds to the 40th percentile among TypeScript projects of similar scale"). The last but not the least, a Say-Do reliability metric will capture the ratio of committed deliverables to actually shipped deliverables per sprint, to formalize measurement of agile execution precision.

VIII. CONCLUSION

This study introduced Trackware, a socio-technical analytics platform designed to address persistent information asymmetry between engineering teams and organizational stakeholders. Trackware provides a unified platform, reducing manual reporting effort, by integrating an EVM translation layer and AI-driven workload modeling that supports predictive project management.

The primary technical contribution is the integration of concurrent data pipelines that convert raw VCS events into business-oriented metrics within a four-second response time. This is complemented by a five-axis health model, presented via interactive visualizations, understandable by both technical and non-technical users. The dual-mode architecture supports GitHub-based tracking as well as manual project-management inputs, which ensures versatile applicability across open-source and proprietary environments, maintaining excellent analytical depth.

Experimental evaluation verified the accuracy of EVM calculations, the reliability of threshold-based alerting, and the latency improvements achieved through concurrent API design. Identified restrictions, including API rate limitations, capped sampling, manual EV-ratio estimation, and uniform health weighting, define a clear trajectory for future work. Planned enhancements such as GraphQL-based acquisition, automated EVM inputs, weighted health scoring, commit-timestamp burnout detection, and contextual benchmarking will make Trackware scalable and a fully automated socio-technical analytics solution.

Overall, Trackware highlights how combining established project-management theory (EVM), modern version-control analytics, and generative AI can yield practical tools that bridge the gap between the domains of engineering and leadership, ensuring aligned operation for both the groups with data-driven visibility.

REFERENCES

- [1] K. Beck et al., "Manifesto for Agile Software Development," Agile Alliance, 2001. [Online]. Available: <https://agilemanifesto.org/>
- [2] Y. Jiang, B. Adams and D. M. German, "Will my patch make it? And how fast? Case study on the Linux kernel," *2013 10th Working Conference on Mining Software Repositories (MSR)*, San Francisco, CA, USA, 2013, pp. 101-110, doi: 10.1109/MSR.2013.6624016.
- [3] Salesforce, "Tableau Pricing", 2026. [Online]. Available: <https://www.tableau.com/pricing>.
- [4] Q. W. Fleming and J. M. Koppelman, *Earned Value Project Management*, 4th ed. Newtown Square, PA, USA: Project Management Institute, 2010.
- [5] W. Lipke, "Schedule is different," *The Measurable News*, vol. 4, pp. 31-34, 2003.
- [6] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open-source software development: Apache and Mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309-346, Jul. 2002.
- [7] G. Gousios, M. Pinzger, and A. van Deursen, "An exploratory study of the pull-based software development model," in *Proc. 36th Int. Conf. on Software Engineering (ICSE)*, Hyderabad, India, 2014, pp. 345-355.
- [8] C. Maslach and M. P. Leiter, *The Truth About Burnout*. San Francisco, CA, USA: Jossey-Bass, 1997.
- [9] M. Claes, M. Mäntylä, J. Kuuttila, and B. Adams, "Do programmers work at night or on weekends?," in *Proc. Int. Conf. on Software Engineering (ICSE)*, 2018, pp. 141-151.
- [10] C. Jones, *Estimating Software Costs: Bringing Realism to Estimating*, 2nd ed. New York, NY, USA: McGraw-Hill, 2007.