


# Inter-Team Dependencies and Delivery Risk: A Network Analysis of Technology Programs

**Ayokunmi Paul Sodamola [1*], Mercy Osato Omoifo-Irefo [2]**
[1] Department of Management Information Systems, Baylor University, Waco, Texas, United States of America
[2] Department of Environmental and Public Health Sciences, University of Cincinnati, Cincinnati, Ohio, United States of America

* Corresponding Author: **Ayokunmi Paul Sodamola**

## Article Info

**ISSN (Online):** 3107-6580
**Volume:** 01
**Issue:** 03
**Received:** 22-04-2025
**Accepted:** 20-05-2025
**Published:** 18-06-2025
**Page No:** 16-27

**Abstract**
**Background:** Large technology initiatives depend on coordinated work across specialized delivery units, creating technical, informational, and scheduling dependencies whose structure may affect delivery performance. Coordination theory and software-engineering research suggest that dependencies generate coordination requirements, but conventional program controls frequently summarize them as isolated risks rather than an interconnected system.
**Purpose:** This study develops and empirically benchmarks a network-based approach for examining whether dependency topology provides information about delivery risk beyond conventional project characteristics.
**Method:** The study uses the TAWOS public Jira dataset. The published version contains 508,963 issues from 44 open-source projects, including 267,568 issue links, while public version 1.1 contains 458,232 issues from 39 projects after five projects were removed. Directed dependency networks are specified from explicit Depend issue links, with software components serving as transparent delivery-unit proxies. Planned analyses include degree, betweenness, density, centralization, concentration, clustering, and modularity, linked prospectively to sprint-window misses and issue-resolution duration.
**Results:** In the real published aggregate benchmark, median issue-link intensity was 0.379 links per issue and varied from 0.064 to 0.824. Link intensity correlated with comments per issue, $\rho = 0.536$, $p < 0.001$, but not conclusively with change-log intensity or bug share. These findings establish marked dependency heterogeneity and an association with coordination activity, but do not establish a centrality-delay effect. Direct longitudinal coefficient estimates are deliberately withheld until the raw network analysis is executed.
**Contribution:** The study formalizes dependencies as network structure rather than isolated entries and provides a reproducible design for testing structural delivery-risk mechanisms.
**Practical implications:** Network indicators could complement existing delivery governance by prioritizing structurally consequential dependencies and potential bottlenecks.

**DOI:** https://doi.org/10.54660/.IJECA.2026.1.1.33-44

**Keywords:** Inter-Team Dependencies, Delivery Risk, Technology Programs, Dependency Network Analysis, Social Network Analysis, Project Risk Management, Coordination Complexity, Program Management

## Introduction

Technology delivery is increasingly organized around differentiated units that own components, services, platforms, features, infrastructure, and integration responsibilities. Differentiation alone does not create complexity. The difficulty arises when those specialized units must exchange artifacts, decisions, information, resources, or completed work in order to deliver an integrated outcome. Coordination theory consequently defines coordination around the management of dependencies among activities, while project-complexity research identifies differentiation and interdependence as foundational components of complexity [3, 4]. In software settings, architecture and organizational arrangements are likewise intertwined because changes crossing

subsystem or organizational boundaries create coordination requirements [5].

This problem is especially consequential in multi-team technology delivery. Dependencies may be sequential, such as one team waiting for another team's API. They may be reciprocal, such as two components requiring simultaneous interface changes. They may involve shared environments, architectural approvals, release sequencing, data contracts, security decisions, or scarce specialists. Empirical software-engineering studies have repeatedly shown that coordination requirements can extend beyond formal team boundaries and that better alignment between required and actual coordination is associated with shorter development time [6]. Distributed work has also been observed to take substantially longer than comparable colocated work in some empirical settings, with the number of people involved strongly associated with completion time [7]. Agile coordination research similarly identifies synchronization, structural arrangements, and boundary spanning as central coordination mechanisms [8].

The central research problem is that conventional delivery governance can acknowledge dependencies without representing their topology. A risk register can record that Team A depends on Team B, and a dependency log can list hundreds of such relationships, yet neither representation necessarily reveals whether one unit lies on many dependency paths, whether dependencies are becoming concentrated around a small subset of units, whether tightly coupled clusters are emerging, or whether a delayed unit has unusually broad downstream exposure. Network analysis addresses this representational problem by considering all nodes and ties simultaneously [9, 10]. Sosa, Eppinger, and Rowles showed that design interfaces and organizational interactions can be analyzed jointly as networks, while Gokpinar, Hopp, and Iravani found that structural position and architecture-organization misalignment were associated with product-quality outcomes in complex development [9, 10].

Software-project risk research provides a complementary foundation. International Delphi research has documented a broad set of recurring software-project risks [11]. Wallace, Keil, and Rai showed that social, technical, and project-management risk dimensions are related to project performance rather than behaving as isolated phenomena [12]. More general project research has found value in systematic risk-management practices while also showing that their use and effectiveness vary with project characteristics and technological uncertainty [13]. These traditions establish the importance of risk, but they do not fully specify how the *configuration* of cross-unit dependencies creates exposure, bottlenecks, and propagation pathways [11-13].

The project-complexity literature strengthens the theoretical bridge. Baccarini conceptualized project complexity in terms of differentiation and interdependency [4]. Williams emphasized structural complexity through the number and interdependence of project elements [15], while Bosch-Rekveldt and colleagues broadened project complexity into technical, organizational, and environmental dimensions [14]. A network representation is therefore not an arbitrary visualization. It is a direct operationalization of one of the mechanisms that complexity theory identifies as central, namely the pattern of interconnections among project elements [4, 14, 15].

The resulting gap has three dimensions. The theoretical gap is insufficient integration between coordination and dependency theory, network structure, and technology-delivery risk. The methodological gap is the continued reliance on counts, matrices, logs, and aggregate complexity indicators that cannot by themselves distinguish peripheral dependencies from brokerage, concentration, community structure, or systemic connectivity. The managerial gap is the absence of interpretable early-warning metrics showing not only how many dependencies exist but where they are structurally concentrated and which delivery units can transmit disruption [3, 6, 9, 10].

The study therefore has the following aim: to investigate how the structural characteristics of inter-unit dependency networks are associated with delivery risk in complex technology delivery, and to determine whether network indicators add information beyond conventional delivery characteristics.

The research addresses five questions:

**RQ1.** What structural characteristics characterize dependency networks in large software and technology-delivery settings?

**RQ2.** How are unit-level network positions associated with subsequent delivery risk?

**RQ3.** How are program or project-level network characteristics associated with aggregate delivery performance?

**RQ4.** Which network measures provide the strongest and most stable indicators of elevated delivery risk?

**RQ5.** Do network measures provide incremental explanatory or predictive value beyond conventional project, issue, and delivery controls?

The hypotheses follow from the edge orientation adopted in this study. A directed edge $A \rightarrow B$ means that delivery unit A depends on an artifact or capability supplied by B. Under this convention, A's out-degree represents upstream dependency exposure, while A's in-degree represents the number of downstream units relying on A. This definition prevents a common ambiguity in interpreting directed dependency networks.

**H1:** Higher total dependency degree is associated with greater subsequent delivery risk.

**H2:** Higher betweenness centrality is associated with greater subsequent delivery risk because brokerage positions can become coordination bottlenecks.

**H3a:** Higher out-degree is associated with greater focal-unit delay risk through greater exposure to upstream providers.

**H3b:** Higher in-degree is associated with greater downstream propagation potential because more delivery units rely on the focal unit.

**H4:** Greater program-level dependency centralization and dependency concentration are associated with greater delivery instability.

**H5:** Density has a nonlinear relationship with delivery risk. Moderate connectivity may support integration, whereas high connectivity may raise coordination burden.

**H6:** Greater modularity is associated with lower cross-unit risk propagation when meaningful modules reduce unnecessary cross-boundary dependencies.

The paper contributes theoretically by treating delivery risk partly as an emergent property of dependency topology rather than only an attribute of individual work items or teams. Methodologically, it specifies directed temporal dependency networks and distinguishes node exposure, brokerage,

concentration, and modularity. Managerially, it proposes leading structural indicators that can complement lagging indicators such as missed milestones, aged blockers, and actual schedule variance [6, 9, 10].

## Literature Review and Theoretical Framework

### Inter-Team Dependencies and Coordination Complexity

Malone and Crowston's coordination theory begins from the proposition that coordination concerns the management of dependencies between activities [3]. This framing is particularly applicable to technology programs because work is decomposed for specialization while final value depends upon reintegration. A dependency therefore has two simultaneous meanings. It is a technical or work relationship, and it creates a coordination requirement. When dependencies proliferate, the number of possible synchronization points, decisions, handoffs, and failure pathways can increase even when every constituent team performs competently [3, 4].

Software development provides unusually observable examples. Herbsleb and Grinter documented how distributed architecture and organizational distance can create severe communication and integration problems [5]. Cataldo and colleagues operationalized coordination requirements from technical task dependencies and found that they were volatile and often crossed organizational boundaries [6]. Their empirical evidence further indicated that greater congruence between required coordination and actual coordination was associated with reduced development time [6]. The implication for technology programs is that delivery risk can arise not merely from the presence of difficult tasks, but from whether the organization can successfully service the coordination demands generated by their dependency structure. [13]

Research on global software development reinforces this mechanism. Herbsleb and Mockus found that distributed work items in their study took approximately two and a half times as long as comparable colocated work items, and that distributed items involved more people, with involvement itself strongly related to calendar completion time [7]. Strode and colleagues' agile cases subsequently described coordination through synchronization, structure, and boundary spanning [8]. Together, these studies indicate that coordination load is not incidental administrative overhead. It is connected directly to how technical work is structured and distributed [7, 8].

### Technical Architecture, Organizational Networks, and Performance

Product-development research provides another important bridge. Sosa, Eppinger, and Rowles represented product interfaces and organizational interactions as networks and examined where expected technical communication and observed team interactions were misaligned [9]. Their results showed that boundaries, interface strength, indirect interactions, and system modularity affected alignment between technical and organizational structures [9]. This establishes a mechanism through which technical interdependence can generate organizational coordination demands.

Gokpinar, Hopp, and Iravani extended this reasoning by relating network characteristics to quality performance in automotive product development [10]. Their analysis found a nonlinear association between centrality in a product-architecture network and warranty-repair outcomes, and showed that a coordination-deficit measure capturing architecture-organization mismatch was positively associated with quality problems [10]. These findings caution against the simplistic assumption that every form of centrality is monotonically harmful. They support testing nonlinearities and distinguishing structural exposure from actual coordination capability [10].

Sosa, Gargiulo, and Rowles later examined whether informal communication networks can themselves disrupt coordination in new-product-development projects [25]. This broader stream of evidence suggests that both insufficient connections and excessive or poorly structured connections may matter. Accordingly, density should not be treated automatically as a risk factor. A nonlinear or threshold specification is theoretically preferable to a one-directional hypothesis [10, 25].

### Delivery Risk and Project Complexity

Delivery risk is defined here as the *ex ante probability or exposure to failure to satisfy a planned delivery commitment*. Realized delay is an outcome through which that risk becomes observable. This distinction matters because a late item is not itself a probability. The empirical analysis therefore uses realized outcomes, such as completion after a planned sprint boundary and unusually long resolution duration, as indicators through which prior structural risk can be evaluated. Software-project-risk research has long distinguished multiple social, technical, and management dimensions affecting performance [11, 12, 24].

Risk-register approaches remain useful for identification and accountability, but relational risks differ from independent event lists because one dependency can alter another unit's exposure. Project-complexity research is explicit that interdependency is itself part of complexity [4, 14, 15]. Thus, two programs with the same number of dependencies can have different risk structures. One may distribute dependencies evenly across modular clusters, while another may route most integration paths through two bottleneck units. A dependency count cannot distinguish those cases. Network centralization, betweenness, modularity, and concentration can [16-19].

### Network Constructs

Centrality measures capture different structural roles and must therefore not be treated as interchangeable. Degree centrality measures direct connectivity [16]. Betweenness captures the extent to which a node lies on shortest paths connecting other nodes and can be computed efficiently using Brandes' algorithm [17]. In a dependency setting, high betweenness identifies delivery units positioned between otherwise more weakly connected parts of the program, making it a plausible indicator of brokerage and integration bottlenecks [17].

At the network level, density measures realized ties relative to possible ties. Centralization measures the extent to which connectivity is organized around a small number of nodes. Modularity quantifies the extent to which a network can be partitioned into communities with more within-community connectivity than expected under a corresponding null structure [18]. Community-detection methods such as the

Louvain approach optimize modularity efficiently in large networks [19]. For program management, modularity is theoretically relevant because a highly modular dependency architecture may contain disruptions within workstreams, while high cross-module coupling can increase propagation opportunities [18, 19].

**Conceptual Model**

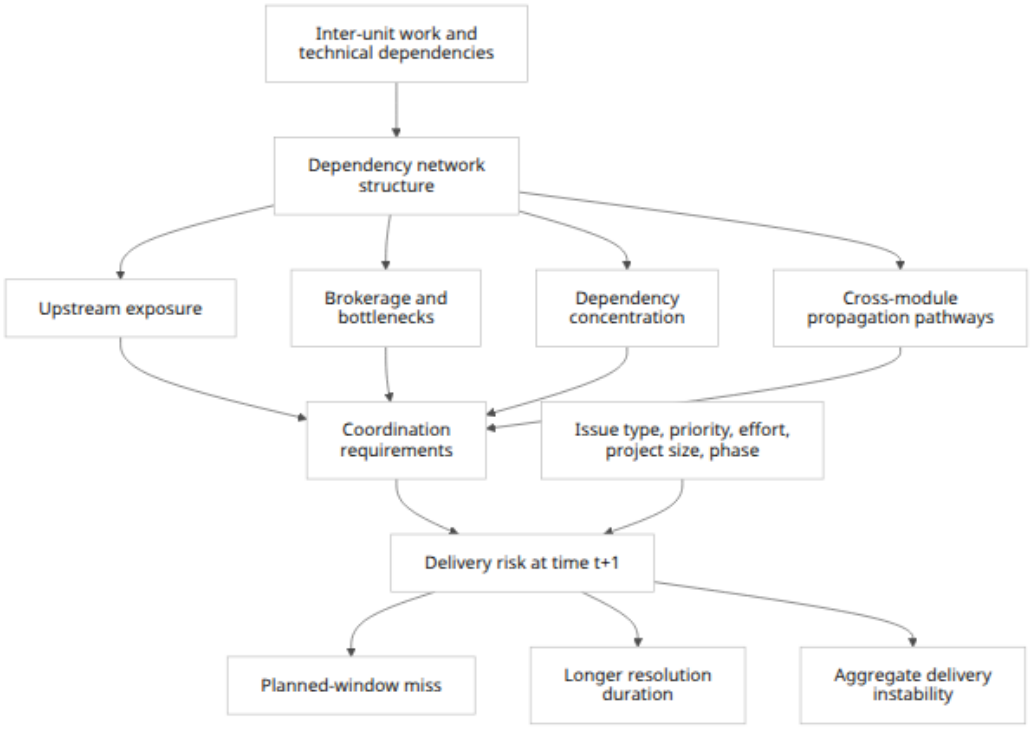



**Fig 1:** Dependency structure is theorized to affect delivery outcomes through exposure, brokerage, concentration, and propagation. Controls prevent structural effects from being conflated mechanically with project size or issue complexity [3, 6, 9, 10].

## Research Methodology

### Research Design

The proposed empirical design is quantitative, observational, longitudinal, and network-analytic. Temporal ordering is essential. Network characteristics measured during period $t$ should predict delivery outcomes during $t + 1$, rather than calculating centrality from information created after the outcome. This reduces temporal leakage and makes the language of risk more defensible. Cross-sectional models should be retained only as descriptive robustness checks, not treated as prospective evidence.

The real-data source is TAWOS. The 2022 peer-reviewed dataset paper introduced a relational Jira database containing issues, comments, change logs, components, sprints, versions, users, and issue links [1]. The original publication covered 44 projects and 508,963 issues. Public TAWOS version 1.1 now contains 39 projects and 458,232 issues across 12 repositories because one repository containing five projects was removed after it disappeared from the public domain [2]. The database is stored as a MySQL relational dump and is licensed under Apache License 2.0 subject to research-use and ethical conditions described by the maintainers [2].

TAWOS is unusually suitable for this study because its Issue_Links relation records source and target issue IDs, relationship names, descriptions, and direction, including explicit Depend relationships [2]. Its Component table supplies a many-to-many mapping between issues and software components. Sprint records contain start, end, activation, and completion dates. Issue records contain creation and resolution dates and Resolution_Time_Minutes, which version 1.1 defines as the interval between an issue's first transition to In Progress and its transition to Resolved [2]. These fields make a temporally ordered dependency and delivery analysis technically possible.

A construct-validity limitation must nevertheless be explicit. TAWOS does not provide verified organizational team membership. Components are therefore treated as delivery-unit proxies, not asserted to be literal teams. A stronger enterprise replication should substitute genuine team IDs while retaining the same graph construction and outcome definitions. This distinction is critical if the final paper retains the phrase “inter-team” in its title.

### Units of Analysis

Three levels are distinguished. At the dyadic level, the observation is a directed dependency between delivery units. At the delivery-unit level, the observation is unit $i$ during temporal window $t$. At the project level, the observation is the complete dependency network for project $p$ during window $t$. Separating levels prevents node metrics, network metrics, and project outcomes from being placed incorrectly into the same statistical model.

### Network Construction

A raw issue dependency is normalized so that $A \rightarrow B$ means “the delivery represented by A depends on B.” Because Jira can store mirrored or directionally described links, direction

must be normalized from Name, Description, and Direction, followed by an audit of all distinct dependency-link labels before analysis [2]. Duplicate mirror records must be collapsed after semantic normalization. Only explicit Depend relations enter the primary network. Broader link types, such as supersedes or duplicates, belong in sensitivity analyses because they do not necessarily represent delivery dependency [2].

Each issue is mapped to its recorded component or components. When an issue has one component, its component assignment is deterministic. When an issue belongs to $m$ source components and the target issue belongs to $n$ target components, the recommended weighted specification assigns $1/(mn)$ of that dependency to each resulting component pair. This fractional rule preserves total edge weight while avoiding artificial multiplication of dependency volume. A robustness model may instead retain only issues with one component.

The network for each project and time window is then represented as $G_{pt} = (V_{pt}, E_{pt}, W_{pt})$, where $V$ denotes delivery-unit proxies, $E$ denotes normalized directed dependency relationships, and $W$ denotes dependency weights. Windows should preferably correspond to sprints where sprint metadata are sufficiently complete. Monthly windows provide a common-calendar sensitivity analysis. Static lifetime networks may be reported descriptively but should not be used for claims that structure precedes delivery risk.

**Table 1:** Network measures and operational interpretation

| Measure | Level | Formal interpretation | Technology-delivery interpretation | Direction tested |
|---|---|---|---|---|
| Out-degree | Unit | Number of outgoing adjacent units | Number of upstream providers on which focal unit depends | Positive with focal delay |
| In-degree | Unit | Number of incoming adjacent units | Number of downstream units relying on focal unit | Positive with propagation |
| Weighted degree | Unit | Sum of edge weights | Dependency load rather than unique-neighbor count | Positive |
| Betweenness | Unit | Share of shortest paths traversing node | Brokerage or integration bottleneck | Positive |
| PageRank or eigenvector score | Unit | Connectivity to structurally influential nodes | Exposure to influential dependency units | Exploratory |
| Density | Network | Observed directed edges divided by possible edges | Overall interconnectedness | Nonlinear |
| Centralization | Network | Concentration of node centrality | Reliance on a small structural core | Positive |
| Dependency HHI | Network | Sum of squared shares of dependency load | Concentration around few units | Positive |
| Clustering | Network | Local tendency to form connected neighborhoods | Local coordination concentration | Exploratory |
| Modularity | Network | Excess within-community connectivity | Structural separation into modules | Negative with propagation |

Degree and betweenness are long-established centrality concepts [16, 17]. Modularity captures network community structure and provides a natural basis for testing whether more modular dependency architectures contain cross-unit propagation [18, 19].

**Outcome Measurement**

The primary direct delivery outcome is a binary planned-window miss for issues assigned to a sprint:

$$Late_i = I(ResolutionDate_i > SprintEndDate_i)$$

for issues with valid sprint and resolution timestamps. This outcome distinguishes planned timing from actual completion. A sensitivity outcome uses sprint Complete_Date - End_Date to quantify sprint-level schedule overrun when those fields are valid.

The principal continuous secondary outcome is log1p(Resolution_Time_Minutes). This field captures active resolution duration in TAWOS version 1.1 and therefore offers a direct development-time measure [2]. Prior software research demonstrates that delivery capability and bug-fix time can be modeled from issue-management histories, supporting the broader validity of time-based software-delivery outcomes [20, 21].

A third outcome is downstream propagation. For a focal unit delayed during $t$, the analysis tests whether units with incoming dependency relationships from the focal unit experience elevated late-delivery incidence during $t + 1$. This moves beyond asking whether central teams are individually slower and addresses the systemic-risk mechanism central to the paper.

**Covariates**

Issue-level controls should include issue type, priority, story points where available, creation period, and relevant workflow attributes. Unit-period controls should include issue volume, active assignee count, historical delay rate, dependency volume, and project phase proxies. Project-level controls should include project size and temporal fixed effects. Controls are included because theory requires them, not because they happen to be available.

**Statistical Analysis**

Descriptive analysis first reports project size, issue counts, dependency-link prevalence, number of components, and temporal coverage. Network analysis then reports distributions of degree, weighted degree, betweenness, density, concentration, components, and modularity. Highly skewed centrality measures are log-transformed where appropriate and standardized within the estimation sample.

The primary inferential model is a multilevel logistic specification:

$$logitP(Late_{i,t+1} = 1) = \beta_0 + \beta_1 Z(OutDegree_{u,t}) + \beta_2 Z(InDegree_{u,t}) + \beta_3 Z(Betweenness_{u,t}) + \gamma X_{i,t} + \alpha_p + \delta_t$$

where $u$ is the issue's delivery unit, $p$ is project, $\alpha_p$ is a project random intercept or project effect, and $\delta_t$ represents time effects. An analogous generalized linear mixed model estimates resolution duration.

Because program-level sample size is limited, project-network models should be parsimonious. Density, centralization, modularity, and dependency concentration should not all be entered simultaneously into an unrestricted regression with only 39 projects. The preferable strategy is a small number of theory-driven specifications, repeated over temporal windows, with project-level clustering or partial pooling.

Incremental value is tested by comparing three models. Model A contains conventional issue and project controls. Model B adds unit history and workload. Model C adds network measures. Explanatory improvement is assessed by likelihood-based fit and effect sizes. Predictive improvement requires genuine out-of-sample validation. Leave-one-project-out validation is preferable to random row splitting because random splitting could allow closely related records from the same project to enter both training and test sets.

For binary outcomes, report discrimination, calibration, Brier score, sensitivity, specificity, precision, and recall where appropriate. AUC alone is insufficient. For continuous resolution duration, report MAE and RMSE on held-out projects. All predictive preprocessing, including standardization, must be fitted inside the training fold.

**Robustness Analyses**

Robustness tests should vary the dependency definition, retain only one-component issues, vary temporal windows, compare weighted and unweighted graphs, omit extreme-centrality observations, restrict to projects with substantial sprint coverage, and compare linear with nonlinear density effects. A version 1.1 analysis should also be compared with results obtainable from the original 44-project benchmark where variables overlap.

**Missing Data**

Missingness is summarized by variable and project before modeling. Issues without components can enter project-level descriptive analyses but cannot support component-network effects. Issues without sprint assignment cannot enter planned-window models but may enter resolution-time models. Story points should not be imputed mechanically for issues never estimated. Multiple imputation, if used for defensible covariates, must occur inside cross-validation folds to prevent information leakage.

**Software and Reproducibility**

Network construction can be implemented in Python using relational extraction plus a network library, or in R using igraph. Mixed models can be estimated in R or Python. The exact interpreter, database, package versions, random seeds, and environment lock file should be archived with the final analysis. A reproducibility script for the aggregate benchmark is supplied with this report.



**Fig 2:** Study design




**Fig 3:** Analysis timeline

The dates above are an analysis-plan illustration, not a promise of future work or evidence that analyses have already occurred.

**Results, Tables, and Figures**

**Sample Characteristics and Validation of the Real Dataset**

The published TAWOS version 1 benchmark contains 44 projects, 508,963 issues, 237,594 bugs, 208,811 users, 6,313 developers, 10,023,871 change-log records, 1,612,399 comments, 267,568 issue links, 2,232 components, 5,029 sprints, 7,885 versions, and 69,724 issues with reported story points [1]. The transcription used for this report was programmatically validated against every published total. The five Talendforge projects contain 50,731 issues. Removing them yields 458,232 issues, exactly reproducing the public version 1.1 issue total [2].

**Table 2:** Dataset characteristics

| Characteristic | Published TAWOS v1 benchmark | Public TAWOS v1.1 |
|---|---|---|
| Jira repositories | 13 | 12 |
| Projects | 44 | 39 |
| Issues | 508,963 | 458,232 |
| Bugs | 237,594 | 215,570 derived after excluding removed repository |
| Issue links | 267,568 | 246,576 derived after excluding removed repository |
| Components | 2,232 | 2,028 derived after excluding removed repository |
| Comments | 1,612,399 | 1,518,327 derived after excluding removed repository |
| Change-log records | 10,023,871 | 9,253,418 derived after excluding removed repository |
| Sprints with published counts | 5,029 | 4,594 derived after excluding removed repository |

***Source:*** published TAWOS Table 1 for version 1 totals [1]. Version 1.1 repository metadata confirm 39 projects and 458,232 issues [2]. Derived version 1.1-compatible counts above simply subtract the five removed Talendforge projects from the published table and should not be confused with a fresh query of the raw version 1.1 database.

**Dependency-Link Heterogeneity**

Recorded issue-link intensity varied markedly across projects. The mean was 0.402 links per issue, the median was 0.379, the standard deviation was 0.182, and the observed range was 0.064 to 0.824. MongoDB Core Server had the highest published link intensity at 0.824, followed by Moodle at 0.784, LSST Data Management at 0.744, Indy Node at 0.701, and Appcelerator Studio at 0.678. These values establish substantial variation in relational workload across projects, although issue links include more than dependency semantics and therefore cannot themselves be interpreted as dependency-network density [1]. Figure 4. Published TAWOS v1 project count by Jira repository.

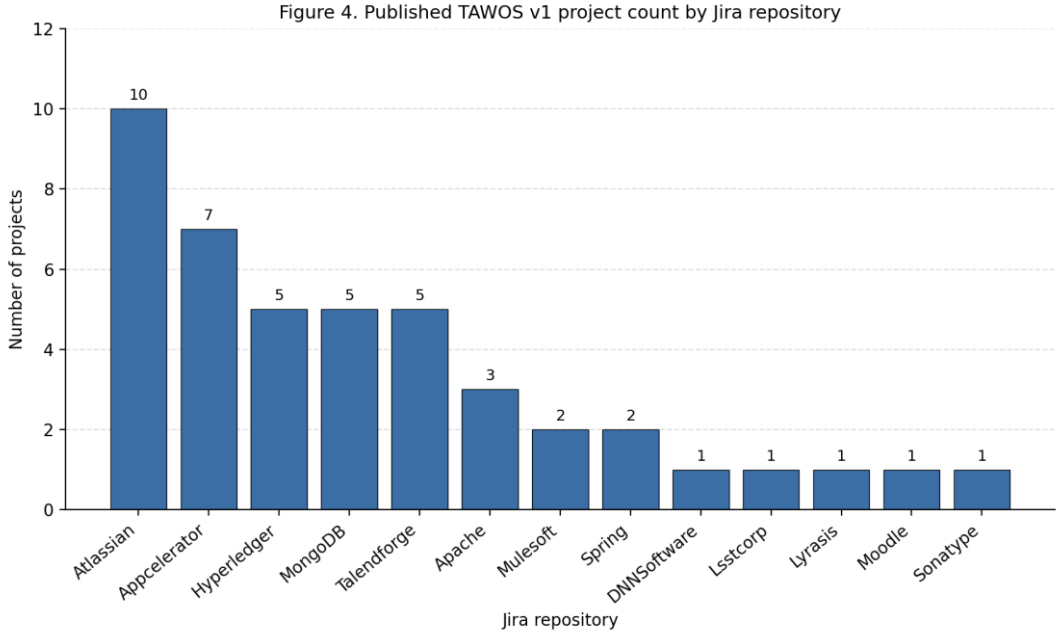



**Fig 4:** Published TAWOS v1 project count by Jira repository The published benchmark spans 13 repositories. Atlassian contributes ten projects, Appcelerator seven, and Hyperledger, MongoDB, and Talendforge five each. Source: author calculations from the published TAWOS Table 1 [1].

Figure 4 also illustrates an external-validity issue. The observations are heterogeneous open-source projects, not a balanced random sample of enterprise technology programs. Project-level inference must therefore include project controls and, where prediction is attempted, must validate across rather than merely within projects [1].

Figure 5. Project size and recorded issue-link volume

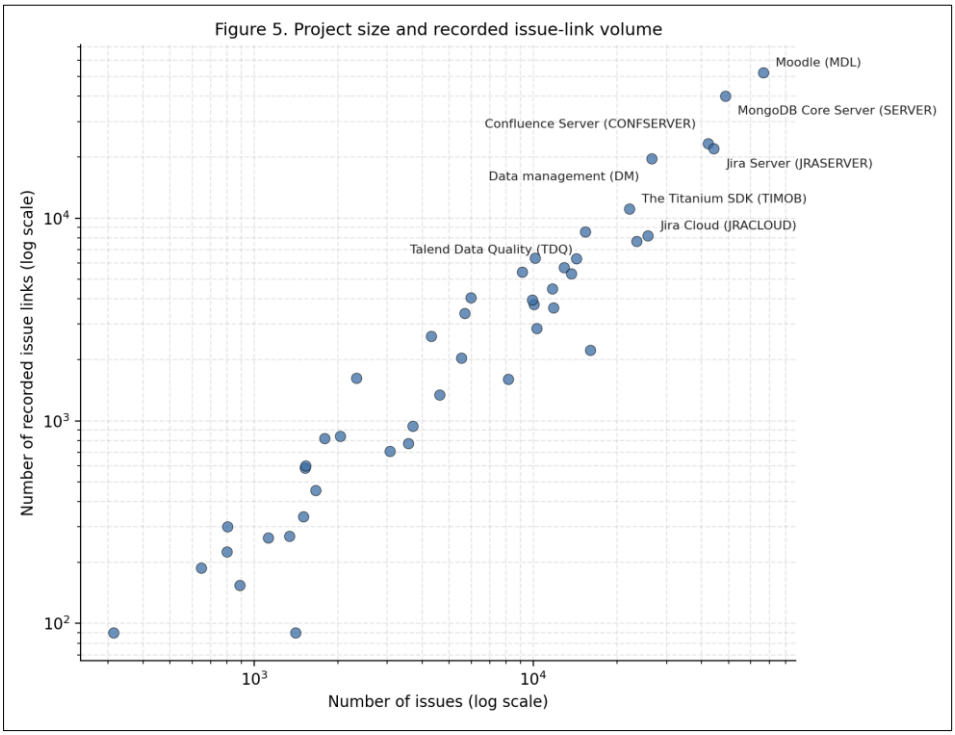



**Fig 5:** Project size and recorded issue-link volume in TAWOS. Both axes are logarithmic. The plot demonstrates that link volume grows substantially with project size while retaining pronounced between-project heterogeneity. Labels identify the eight projects with the largest absolute numbers of recorded links. Source: author calculations from published TAWOS Table 1 [1].

The relationship in Figure 4 is one reason that absolute dependency counts should not be interpreted without scale adjustment. A large project can have many links simply because it contains many issues. Link intensity, network density, standardized degree, and model controls for project size are therefore needed before structural exposure is interpreted.

Figure 6. Projects with the highest recorded issue-link intensity

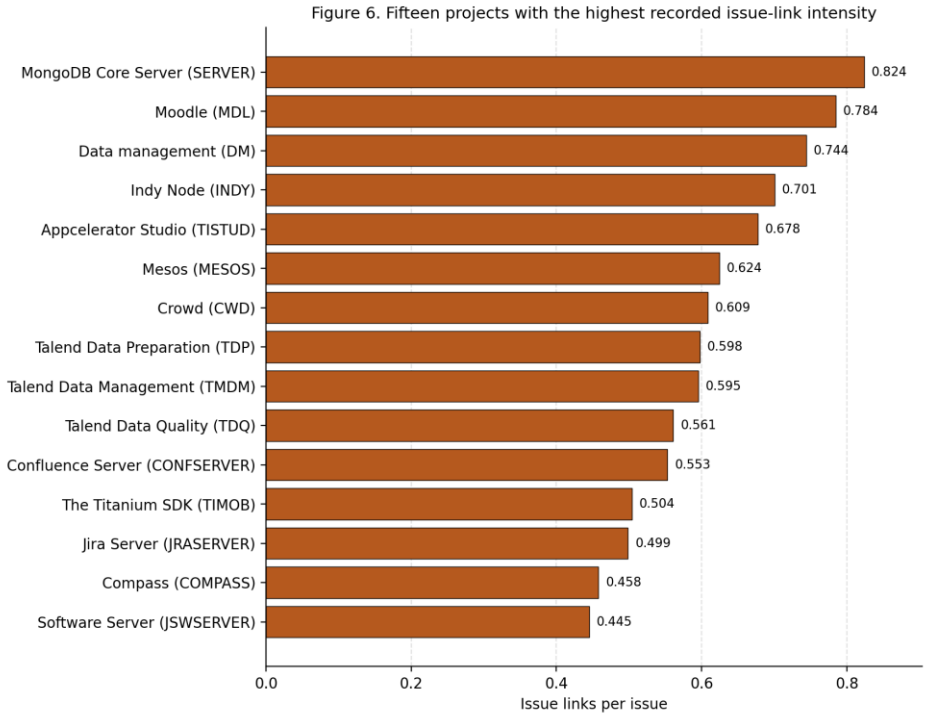



**Fig 6:** Fifteen projects with the highest published issue-link intensity. The metric is the number of recorded issue links divided by the number of issues. It is an aggregate relational-intensity indicator, not a team-network centrality measure. Source: author calculations from published TAWOS Table 1 [1].

**Exploratory Association Analysis**

To determine whether the aggregate dependency signal is at least consistent with increased coordination activity, links per issue were correlated with comments per issue and change-log events per issue. Bug share was included as a distinct development-burden indicator. Spearman correlations were used because the project distributions are heterogeneous and strongly skewed.

**Table 3:** Real-data exploratory associations in the 44-project published benchmark

| Predictor | Outcome | Spearman $\rho$ | p-value | n | Interpretation |
|---|---|---|---|---|---|
| Links per issue | Comments per issue | **0.536** | **0.000176** | 44 | Moderate positive association |
| Links per issue | Change-log events per issue | 0.278 | 0.067445 | 44 | Positive but inconclusive |
| Links per issue | Bug share | 0.270 | 0.076256 | 44 | Positive but inconclusive |

The significant association between link intensity and comment intensity is consistent with the theoretical expectation that more relationally connected work creates greater coordination activity [3, 6]. It should not be overinterpreted. Comments are only a proxy for observable discussion, recorded links include multiple link semantics, and a cross-project correlation cannot establish temporal order or causality. The result is best interpreted as construct-supporting preliminary evidence that relational workload and observable coordination activity covary [3, 6].

A sensitivity analysis excluding all five Talendforge projects produces a similar correlation between link intensity and comments per issue, $\rho = 0.566$, $p = 0.000175$, $n = 39$. The correlations with change-log intensity, $\rho = 0.231$, $p = 0.158$, and bug share, $\rho = 0.251$, $p = 0.123$, remain inconclusive. This stability reduces concern that the principal aggregate coordination association is driven solely by the repository removed between TAWOS versions.

**Table 4:** Separation of completed evidence from execution-required confirmatory tests

| Hypothesis | Required evidence | Status in this report | Permissible claim |
|---|---|---|---|
| H1, degree and delivery risk | Raw directed dependency network plus subsequent delivery outcome | Not yet estimated | No coefficient claimed |
| H2, betweenness and risk | Raw adjacency-level dependency paths | Not yet estimated | No coefficient claimed |
| H3a, out-degree exposure | Direction-normalized dependencies plus subsequent focal delay | Not yet estimated | No odds ratio claimed |
| H3b, in-degree propagation | Direction-normalized dependencies plus downstream outcomes | Not yet estimated | No propagation coefficient claimed |
| H4, concentration and instability | Repeated project-network windows and delivery outcomes | Not yet estimated | No causal claim |
| H5, nonlinear density | Repeated network density with temporal outcomes | Not yet estimated | Nonlinearity remains hypothesis |
| H6, modularity | Raw component networks and cross-module outcomes | Not yet estimated | No protective effect claimed |
| Preliminary construct check | Published aggregate link and coordination data | Completed | Link intensity covaries with comment intensity |

This table is intentionally different from a fabricated regression table. A publication claiming, for example, that "betweenness increased delay odds by 35%" without having computed that model from the raw adjacency data would constitute invented evidence. The correct scientific result at the present stage is that the aggregate benchmark supports the plausibility and feasibility of the network construct, while the principal hypotheses remain to be estimated from version 1.1 or a genuine enterprise program dataset.

**Discussion, Conclusion, and Declarations**

**Principal Findings**

The evidence supports three conclusions at different levels of strength. First, the public software-delivery dataset clearly contains enough relational and temporal structure to support a direct dependency-network study. TAWOS version 1.1 exposes explicit dependency links, components, sprint timing, workflow history, and resolution duration [2]. Second, the published aggregate data demonstrate pronounced between-project heterogeneity in issue-link intensity. Third, higher aggregate link intensity is associated with higher comment intensity, consistent with the coordination-theory proposition that dependencies generate coordination requirements [3, 6]. What has **not** yet been demonstrated is the paper's strongest intended claim, namely that specific network positions prospectively predict schedule delay or that program-level topology predicts delivery instability.

The aggregate association is theoretically meaningful but deliberately modest. Project-level comments can arise for many reasons, including issue complexity, community norms, project age, or contributor structure. Likewise, issue links can represent duplicate, superseding, or other relationships unless explicit Depend semantics are selected [2]. The raw analysis is therefore necessary because it can restrict edges to dependencies, establish direction, map them to delivery units, and ensure that network predictors occur before delivery outcomes.

**Team-Level Network Effects**

Under the specified edge convention, out-degree captures how many upstream units a focal delivery unit depends upon. High out-degree can expose the focal unit to more external completion conditions, approvals, or artifacts. In-degree captures how many other units depend upon the focal unit. High in-degree therefore represents potential propagation reach rather than necessarily the focal unit's own exposure. Distinguishing these mechanisms improves upon an undifferentiated "dependency count" and permits a stronger

test of whether teams are risky because they wait on many others, because many others wait on them, or both.

Betweenness offers a different mechanism. A delivery unit can have only moderate degree yet occupy paths connecting otherwise weakly coupled clusters. Such brokerage roles have particular managerial significance because an integration team, shared platform, security group, release-management function, or common data service may connect multiple workstreams. Betweenness is therefore theoretically related to coordination bottlenecks rather than simply workload [17].

The relationship may also be nonlinear. The finding by Gokpinar and colleagues that product-network centrality related nonlinearly to quality problems warns against interpreting all additional connectivity as harmful [10]. Some connectivity is necessary for knowledge flow and system integration. The relevant question is whether structural demand exceeds coordination capacity, or whether a small number of nodes accumulate disproportionate dependency responsibility [10].

**Program-Level Topology**

Program-level topology turns delivery risk into a systemic question. Two programs can contain the same number of teams and the same number of dependencies while exhibiting different centralization, modularity, and bottleneck structures. A highly centralized graph concentrates dependency relationships around a structural core. A modular graph, by contrast, can localize many interactions within workstreams while limiting cross-module interfaces. Community structure and modularity therefore provide a formal way to test the frequently asserted managerial value of architectural decoupling [18, 19].

This perspective does not imply that dependencies should simply be minimized. Some dependencies encode essential integration. Removing them on paper does not remove the underlying technical relationship. The managerial objective is instead to identify dependencies whose structural importance is disproportionate, reduce accidental coupling, create reliable interfaces, add capacity or redundancy around critical providers, and shorten unresolved dependency duration.

**Theoretical Contribution**

The first theoretical contribution is a structural reconceptualization of delivery risk. Existing software-risk research demonstrates that technical, social, and management risks interact [11, 12]. The present framework extends that logic by proposing that risk also resides in the configuration of relationships among delivery units. This turns dependency from a risk-register attribute into a property of a connected system [11, 12].

The second contribution separates exposure from propagation. A delivery unit with high out-degree is exposed to many providers, while a unit with high in-degree can affect many downstream consumers. Betweenness captures a third role, brokerage. This decomposition creates more precise mechanisms than a single "number of dependencies" variable.

The third contribution connects program architecture with delivery governance. Product-development research has already demonstrated that the alignment of technical and organizational structures can matter for performance [9, 10]. Applying comparable network logic to temporally observed technology delivery extends that literature from product quality and communication toward schedule and delivery-risk outcomes [9, 10].

Figure 7 Managerial implications

A practical network-based early-warning process follows directly:




**Fig 7**

Program managers could incorporate such a process into existing dependency reviews rather than replace them. The value added by network analysis is prioritization. An unresolved dependency with ordinary severity but extreme betweenness or broad downstream reach may warrant earlier intervention than an isolated dependency with similar local severity. Conversely, high raw dependency counts may be less concerning when they occur inside a stable, well-coordinated module.

Structural indicators can also inform architecture and staffing. Repeated concentration around one shared platform team may indicate the need for self-service capabilities, delegated ownership, additional capacity, better APIs, or clearer service-level commitments. Persistent cross-community dependencies may reveal architectural boundaries that do not correspond well to delivery ownership. This is consistent with prior evidence that technical interfaces and organizational interactions need to be considered jointly [9, 10].

**Implications for Technology-Program Governance**

Dependency governance should assign explicit ownership not only to individual dependencies but also to structurally critical nodes. A program dashboard could report unresolved dependency count alongside weighted out-degree, in-degree, betweenness percentile, dependency concentration, and trend in cross-module edges. These metrics should be interpreted as diagnostic signals, not performance scores for punishing teams. A central team may be structurally important because the architecture deliberately assigns it shared responsibility.

Governance should therefore distinguish structural risk from team performance. A bottleneck can be generated by architecture, resource policy, organizational design, or shared-services strategy even when the focal team performs efficiently. Treating centrality as poor performance would create the wrong incentive and could encourage teams to suppress dependency reporting. The more appropriate question is whether organizational capacity and escalation mechanisms match structural demand.

**Limitations**

The most important limitation is construct validity. TAWOS components are not verified organizational teams. A manuscript making strong claims about "inter-team" effects should ideally replicate the analysis with true team ownership data. The TAWOS version should describe nodes as component-based delivery-unit proxies throughout its methods and results.

Second, the study is observational. Even a temporally lagged relationship between centrality and subsequent delay does not prove that centrality caused the delay. Difficult architectural work may simultaneously create many dependencies and long completion times. Controls, fixed effects, lagging, and robustness analyses reduce this concern but do not eliminate unobserved confounding.

Third, Jira records formalized dependencies, not every actual dependency. Informal knowledge dependencies, undocumented architectural assumptions, verbal commitments, and coordination occurring outside Jira may be missing. Recorded dependency structure should therefore be understood as an observable subset of the real coordination system [1, 2].

Fourth, open-source projects differ from enterprise technology programs. Governance authority, team stability, confidentiality, release commitments, incentives, and customer obligations may differ substantially. External validity should therefore be tested explicitly using industrial programs before universal managerial claims are made.

Fifth, the aggregate empirical results reported here are not the paper's confirmatory network results. They validate data transcription, demonstrate dependency heterogeneity, and provide an exploratory coordination association. They do not estimate H1-H6. This limitation is preferable to replacing missing evidence with simulated or invented coefficients.

**Future Research**

Future studies should use genuine team-level longitudinal data, distinguish technical, informational, schedule, and resource dependencies, and examine multiplex networks in which the same teams can be connected through several dependency types. Dynamic network models could determine whether increasing centralization or cross-module coupling precedes delivery deterioration rather than merely appearing after problems emerge.

Causal research could exploit organizational or architectural interventions that alter dependency structure. Examples include service decomposition, platform self-service adoption, ownership reorganization, or removal of shared integration gates. Difference-in-differences or other quasi-experimental designs could then test whether structural change produces measurable changes in delivery performance.

Risk propagation also deserves direct study. A particularly valuable extension would estimate whether a delay at unit $j$ increases the hazard of subsequent delay at units structurally dependent on $j$, conditional on issue complexity and historical performance. Such work would move network-based delivery risk from static centrality analysis toward empirical models of cascading disruption.

**Conclusion**

Large technology initiatives should not be conceptualized merely as collections of independently performing teams. Their outcomes also depend on how those teams, components, and workstreams are connected. Coordination theory, project-complexity research, and prior network studies all provide reasons to expect dependency structure to matter [3-10].

The real TAWOS benchmark analyzed here shows substantial heterogeneity in relational intensity and a robust aggregate association between issue-link intensity and observable discussion activity. The public version 1.1 provides the raw relational and temporal fields needed for the more decisive next analysis, namely directed dependency networks linked prospectively to delivery outcomes [1, 2]. The centrality-to-delay hypotheses have not been presented as established findings because doing so before executing that raw analysis would be scientifically indefensible.

The broader significance is nevertheless clear. Dependency management can advance from asking only which dependencies exist to asking where dependencies are concentrated, which delivery units occupy critical structural positions, how disruption can propagate, and whether network structure provides early information about delivery deterioration. A validated network-based approach would complement rather than replace conventional program controls, giving technology leaders a systems-level view of coordination and delivery risk.

**References**


1. Tawosi V, Al-Subaihin A, Moussa R, Sarro F. A versatile dataset of agile open source software projects. Proceedings of the 19th International Conference on Mining Software Repositories. 2022:707-711. doi:10.1145/3524842.3528029
2. SOLAR Group, "The TAWOS Dataset, Version 1.1," 2022, research dataset, doi: 10.5522/04/21308124. The repository documents 458,232 issues from 39 projects in 12 public Jira repositories and the relevant issue-link, sprint, component, and resolution-time fields.
3. Malone TW, Crowston K. The interdisciplinary study of coordination. ACM Computing Surveys. 1994;26(1):87-119. doi:10.1145/174666.174668
4. Baccarini D. The concept of project complexity. A review. International Journal of Project Management. 1996;14(4):201-204. doi:10.1016/0263-7863(95)00093-3
5. Herbsleb JD, Grinter RE. Architectures, coordination, and distance: Conway's Law and beyond. IEEE Software. 1999;16(5):63-70. doi:10.1109/52.795103
6. Cataldo M, Wagstrom PA, Herbsleb JD, Carley KM. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. Proceedings of the ACM Conference on Computer Supported Cooperative Work. 2006:353-362. doi:10.1145/1180875.1180929
7. Herbsleb JD, Mockus A. An empirical study of speed and communication in globally distributed software development. IEEE Transactions on Software Engineering. 2003;29(6):481-494. doi:10.1109/TSE.2003.1205177
8. Strode DE, Huff SL, Hope B, Link S. Coordination in co-located agile software development projects. Journal of Systems and Software. 2012;85(6):1222-1238. doi:10.1016/j.jss.2012.02.017
9. Sosa ME, Eppinger SD, Rowles CM. The misalignment of product architecture and organizational structure in

complex product development. Management Science. 1674;50(12):1674-1689. doi:10.1287/mnsc.1040.0289
10. Gokpinar B, Hopp WJ, Iravani SMR. The impact of misalignment of organizational structure and product architecture on quality in complex product development. Management Science. 2010;56(3):468-484. doi:10.1287/mnsc.1090.1117
11. Schmidt R, Lyytinen K, Keil M, Cule P. Identifying software project risks: An international Delphi study. Journal of Management Information Systems. 2001;17(4):5-36. doi:10.1080/07421222.2001.11045662
12. Wallace L, Keil M, Rai A. How software project risk affects project performance: An investigation of the dimensions of risk and an exploratory model. Decision Sciences. 2004;35(2):289-321. doi:10.1111/j.00117315.2004.02059.x
13. Raz T, Shenhar AJ, Dvir D. Risk management, project success, and technological uncertainty. R&D Management. 2002;32(2):101-109. doi:10.1111/1467-9310.00243
14. Bosch-Rekveldt M, Jongkind Y, Mooi H, Bakker H, Verbraeck A. Grasping project complexity in large engineering projects: The TOE framework. International Journal of Project Management. 2011;29(6):728-739.
15. Williams TM. The need for new paradigms for complex projects. International Journal of Project Management. 1999;17(5):269-273. doi:10.1016/S0263-7863(98)00047-7
16. Freeman LC. Centrality in social networks: Conceptual clarification. Social Networks. 1979;1(3):215-239.
17. Brandes U. A faster algorithm for betweenness centrality. Journal of Mathematical Sociology. 2001;25(2):163-177. doi:10.1080/0022250X.2001.9990249
18. Newman MEJ. Modularity and community structure in networks. Proceedings of the National Academy of Sciences. 2006;103(23):8577-8582. doi:10.1073/pnas.0601602103
19. Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment. 2008. doi:10.1088/1742-5468/2008/10/P10008
20. Choetkiertikul M, Dam HK, Tran T, Ghose A, Grundy J. Predicting delivery capability in iterative software development. IEEE Transactions on Software Engineering. 2018;44(6):551-573.
21. Habayeb M, Murtaza SS, Miranskyy A, Bener AB. On the use of hidden Markov model to predict the time to fix bugs. IEEE Transactions on Software Engineering. 2018;44(12):1224-1244.
22. Ortu M, Destefanis G, Adams B, Murgia A, Marchesi M, Tonelli R. The JIRA repository dataset: Understanding social aspects of software development. Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering. 2015. doi:10.1145/2810146.2810147
23. Dybå T, Dingsøyr T. Empirical studies of agile software development: A systematic review. Information and Software Technology. 2008;50(9):833-859. doi:10.1016/j.infsof.2008.01.006
24. Boehm BW. Software risk management: Principles and practices. IEEE Software. 1991;8(1):32-41.
25. Sosa ME, Gargiulo M, Rowles C. Can informal communication networks disrupt coordination in new product development projects?. Organization Science. 2015;26(4):1059-1078. doi:10.1287/orsc.2015.0974

**How to Cite This Article**

Sodamola AP, Omoifo-Irefo MO. Inter-team dependencies and delivery risk: a network analysis of technology programs. International Journal of Engineering and Computational Applications. 2025;1(3):16–27. doi:10.54660/.IJECA.2026.1.1.33-44.

**Creative Commons (CC) License**

This is an open access journal, and articles are distributed under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) License, which allows others to remix, tweak, and build upon the work non-commercially, as long as appropriate credit is given and the new creations are licensed under the identical terms.