American University Kyiv

A Capstone Project

IMPROVING TASK COMPLETION PREDICTABILITY AND WORK VISIBILITY IN OUTSOURCED IT PROJECTS USING PROCESS FORMALIZATION AND AI – ASSISTED COMMUNICATION AUTOMATION

by **Petro Diak**

Presented in Partial Fulfillment of the Requirements
for the Master
Degree

APPROVED BY:
Volodymyr Vakhitov, Ph.D., Faculty Mentor

2025

ABSTRACT

Outsourced IT projects often experience unpredictable task completion, limited real-time visibility, and coordination inefficiencies caused by inconsistent status-update practices and fragmented communication. In the studied project environment, Data Engineers and QA engineers update work items in Azure DevOps irregularly, follow personal rather than shared reporting routines, and frequently communicate progress through ad hoc messages in Microsoft Teams. As a result, task boards do not reliably reflect the actual state of work, project architects and leads become overloaded with coordination and clarification activities, and project managers lack timely information to support planning and decision-making. These issues ultimately reduce delivery predictability and erode client confidence.

This Capstone addresses these challenges by designing a Task Visibility and Predictability Framework that combines process formalization with AI-assisted communication automation. Using a qualitative research methodology grounded in semi-structured interviews with delivery-team members, the study applies open and axial coding to identify systemic causes of visibility breakdowns, delayed updates, and coordination overload. Drawing on Agile governance literature, project-management best practices, and recent research on AI-supported qualitative analysis, the framework defines clear reporting responsibilities, standardized update routines, and structured communication rules embedded within existing enterprise tools.

The proposed framework introduces formal status-update triggers, minimal documentation standards, and role-specific responsibilities for maintaining task visibility. In addition, lightweight AI-assisted mechanisms—such as rule-based reminders, stale-task detection, and Azure DevOps – Microsoft Teams integration—are used to reinforce timely updates without relying on machine-learning models or extensive historical datasets. This makes the solution practical for real-world implementation under typical organizational and data constraints.

Application of the framework demonstrates improved situational awareness, fewer stale or misleading task states, reduced reliance on synchronous status meetings, and lower coordination burden on architects and technical leads. The findings indicate that even low-complexity automation, when aligned with clearly defined processes, can significantly enhance transparency, predictability, and execution reliability in outsourced IT projects. This Capstone

contributes to a practitioner-oriented model for improving task visibility through workflows and AI-supported communication.

Keywords: task visibility, predictability, Azure DevOps, qualitative analysis, AI-assisted automation, outsourced IT projects, coordination overload

Problems and solutions for them:

Problem: customer ask, “When such task will be resolved?”, “Why such task does not resolve for a long time?”.

Solution: It must be process-driven communication framework with clear status update rules, automated reminders, defined ownership, transparent schedule visualization

Table of Contents

CHAPTER 1: INTRODUCTION, PROBLEM, OBJECTIVES, AND SCOPE

Project management is a structured approach to planning, executing, monitoring, and closing projects to achieve project objectives within next triangle constraints: time, cost, and scope. Decision – making is a fundamental aspect of project management, influencing project success by determining resource allocation, risk mitigation strategies, and overall project direction (Kerzner, 2017).

1.1 Introduction

Projects involving outsourced IT development run in dynamic, dispersed, often high-stress environments where effective delivery depends on open communication, accurate status information, and consistent task completion. Teams working on these projects often span several time zones, work under changing client priorities, and rely on digital tools including Azure DevOps (ADO) and Microsoft Teams for collaboration. Many teams struggle with uneven task updates, unclear ownership of reporting responsibilities, and scattered communication practices even if Agile frameworks are widely embraced. These problems limit a project manager's (PM's) capacity to properly allocate resources, forecast delays, and keep alignment with client expectations, so reducing visibility into real work progress.

The project environment under observation reflects a common scenario of this larger industry problem. Developers and QA engineers lack consistent reporting standards, rely mostly on unofficial communications, and update job statuses in different ways. Work in progress thus gets hidden, decision-making slows down, and the architect meant to be primarily technical gets overwhelmed with coordination and clarification chores. These shortcomings cause delivery risks, team discontent, and client unhappiness.

Recent advances in artificial intelligence (AI) and process automation present pragmatic chances to improve transparency without the need for complex, data-intensive machine-learning systems. Automated reminders, structured prompts, anomaly detection based on timestamps, and team and ADO integration help to support communication routines and guarantee that task status data stays accurate and current by means of lightweight artificial intelligence features. These tools can greatly improve visibility, light coordinators workload, and increase task completion consistency when combined with process formalization.

Designed to improve the accuracy and timeliness of work updates, increase situational awareness, and lower uncertainty in an outsourced IT project, this Capstone project concentrates

on creating a practical solution that integrates process formalization and artificial intelligence – assisted communication automation.

1.2 Background of the problem

Even though Agile practices emphasize transparency, accountability, and continuous communication, real-world delivery environments frequently deviate from these principles. In the outsourced IT project that was observed, the predictability and visibility of task completion were reduced as a result of several operational behaviors:

- Irregular and inconsistent task updates – Developers and QA engineers frequently postpone updating task statuses or provide extremely optimistic or vague ETAs, which leaves the PM uncertain about the actual progress.
- Dependence on informal communication – Team members communicate status through scattered Teams messages, resulting in information loss and version inconsistency, rather than consistently updating ADO.
- Lack of defined reporting expectations – The team's inconsistent reporting habits and misunderstandings regarding responsibilities are the result of the absence of standardized rules regarding the frequency, format, and content of progress updates.
- Project manager uncertainty in allocation decisions – The PM cannot fairly assign new work or evaluate sprint risk without consistent knowledge of when current tasks will be finished.
- Architect overload and burnout risk – The architect spends a lot of time checking tech task status and clarifying expectations with developers, working on projects that should typically be managed via consistent reporting systems.
- Client dissatisfaction due to missed commitments – Delivery delays and ambiguous explanations of slippage erode client trust and pose a reputational risk for the vendor team.

These challenges show that the main problem is not team members' lack of effort or incompetence but rather the **lack of a formalized and automated reporting framework** able of guaranteeing timely, consistent, and honest communication among all stakeholders.
Academic and business research repeatedly shows that formalization of communication practices improves coordination and lowers uncertainty in distributed and outsourced teams. By lowering

manual overhead, pushing team members to update tasks, and offering early warnings when work stalls, AI-assisted automation can also help to reinforce these routines.

1.3 Problem statement

The central problem addressed in this Capstone is:

> Outsourced IT projects lack a formalized, open, and consistent mechanism for reporting task progress and work schedules, so producing unpredictable task completion, inadequate visibility for decision-making, project architect overload, and lower client satisfaction.

This gap shows up in inconsistent progress updates, unclear ownership of reporting duties, reliance on unofficial channels of communication, and lack of automation to spot or stop visibility breakdowns.

1.4 Purpose of the study

The purpose of this study is to design and evaluate a Task Visibility and Predictability Framework that integrates process formalization with AI – assisted communication automation. The framework aims to ensure that task – status information in Azure DevOps is consistent, timely, and actionable, reducing uncertainty and improving the predictability of work completion in outsourced IT projects.

1.5 Research objectives and questions

The main objectives of the Capstone project are as follows:

1. To analyze the root causes of delayed and inconsistent task – status updates in the observed project.
2. To design a formalized communication and reporting process that defines clear rules, intervals, responsibilities, and templates for updating task progress.
3. To implement lightweight AI – assisted automation mechanisms – such as Teams reminders, ADO triggers, and detection of stale tasks – to reinforce the reporting process and enhance visibility.
4. To evaluate how the proposed framework improves task completion predictability, reduces architect overload, and increases transparency.

The main research questions outlined in this paper are:

RQ 1. How does the lack of formalized communication and status – update routines impact task completion predictability in outsourced IT projects?

RQ 2. What process – formalization mechanisms can be introduced to ensure consistent, accurate, and timely task – status updates in Azure DevOps?

RQ 3. How can AI – assisted communication automation (such as Teams reminders, rule – based triggers, and stale – task detection) enhance real – time work visibility?

RQ 4. In what ways does improved transparency reduce coordination overload on key roles such (project architect, etc.)?

RQ 5. What measurable improvements – such as reduced delays, fewer escalations, or more accurate ETAs – can be achieved after implementing a formalized and automated task – visibility framework?

1.6 Scope of the project

This Capstone focuses on improving task progress visibility and predictability within outsourced IT project team. The scope includes:

- **Process analysis** of current communication and reporting behaviors.
- **Design of process formalization elements**, including roles, rules, templates, and routines.
- **Integration of AI – assisted automation** using existing tools: Microsoft Teams, Azure DevOps, and rule – based notifications.
- **Evaluation of the proposed framework** through qualitative assessment, expert judgment, and alignment with best practices from literature.
- **Development of recommendations and implementation guidelines** for long – term adoption.

1.7 Limitations

This study does not include the following:

- Machine-learning or predictive analytics models due to NDA in project, time constraints and lack of comprehensive datasets (project restriction on data sharing)
- Large-scale quantitative evaluation, as the framework is tested in a single project environment

- Client-side interviews or surveys, focusing instead on internal delivery – team interviews and operations
- Tool development or custom software engineering, relying solely on features available within existing enterprise tools
- Organization-wide Agile transformation, as the scope is limited to the operations of a single team

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter reviews key academic and practitioner literature relevant to the problem of insufficient transparency and poor predictability of task completion in outsourced IT projects. This document consolidates research findings on formalization and governance in project management; the utilization of information visualization (information radiators) and progress tracking for agile software teams; communication and coordination in globally distributed software development; and innovative methodologies for automated or tool-assisted progress reporting. The objective is to identify evidence that endorses the implementation of a structured, formalized, and partially automated task – visibility framework.

2.2 Formalization and governance in project and portfolio management

A foundational insight from project management literature is that formalization of processes, decision paths, reporting structures, and communication flows significantly improves project and portfolio success – especially when complexity is high. The seminal study Formalization of Project Portfolio Management: The Moderating Role of Portfolio Complexity by Juliane Teller, Barbara N. Unger, Alexander Kock and Hans-Georg Gemünden (2012) shows that formalization at the project and portfolio levels correlates positively with portfolio performance and success – and that simultaneous formalization at both levels yields a complementary effect, improving outcomes especially in complex portfolios. "Formalization of project portfolio management: The moderating role of project portfolio complexity"

The authors argue that formalization supports "structured information processing and coordination" across projects – ensuring clarity of roles, responsibilities, information flows, and reporting routines. "Formalization of project portfolio management: The moderating role of project portfolio complexity" This aligns with Capstone's premise: that lack of explicit rules and routines for status updates leads to unpredictable task completion and low visibility.

More recently, empirical work on project – portfolio information systems Project Portfolio Management Information Systems' – PPMIS (2020) by Kock et al. found that application of PPMIS is positively associated with quality of portfolio management processes and success – but only when underlying formalization of single projects, portfolio management, and risk management is sufficiently high.

https://www.sciencedirect.com/science/article/abs/pii/S0263786320300314" That suggests that simply using digital tools (or AI – assisted tools) is not enough – their benefit depends on embedding them into a well – formalized process.

Implication for Capstone: formalization is not bureaucracy – it is a success factor. For my aim – level visibility problem, designing a formalized framework (reporting expectations, roles, routines) is theoretically grounded and proven beneficial in complex project environments.

2.3. Agile, information radiators and progress visibility

In software development and Agile contexts, the concept of an Information Radiator is widely endorsed. According to the Agile Alliance, an information radiator is a "visual display… placed in a highly visible location so that all team members … can see the latest information at a glance" – e.g., Kanban boards, burndown charts, or digital dashboards showing test status, build success, backlog state, etc. https://agilealliance.org/glossary/information – radiators

A systematic mapping study, Information Visualization for Agile Software Development (2016), found that visualization techniques help agile teams increase knowledge sharing and awareness of software artifacts and progress across team members. https://www.researchgate.net/publication/287301466_Information_Visualization_for_Agile_Software_Development The study shows that both physical and digital visualizations (cards, boards, dashboards) significantly contribute to coordination, tracking progress, and reducing reliance on verbal or ad – hoc communication.

Given that many agile teams still rely on manual status updates and meetings, the use of interactive, shared visual boards helps externalize progress status in a way accessible to all stakeholders. This reduces information asymmetry and makes delays or blockages visible early. https://www.researchgate.net/publication/287301466_Information_Visualization_for_Agile_Software_Development

Implication for Capstone: If current use of tools (e.g., Azure DevOps, Teams) does not effectively function as an information radiator (due to inconsistent updating or informal communication), then formalizing and structurally embedding a digital information radiator – with enforced update rules – can substantially improve real – time visibility and coordination.

2.4. Progress tracking in distributed and outsourced Agile teams

Globally outsourced projects and distributed software development add even more complexity: time zones, asynchronous communication, lack of face-to-face contact, and cultural diversity. These increase the hazards related to ad hoc status reporting and unofficial communication.

Key work here is the 2013 PhD thesis A Computer – Based Holistic Approach to Managing Progress in Distributed Agile Projects by S. Alyahya, which argues that neither informal (verbal, chat) nor manual formal methods (status meetings, manual reports) are sufficient alone to consistently manage progress in distributed agile projects. Rather, Alyahya suggests a computer-based, holistic approach combining technical activity signals (such as code commits, builds, test results) with hand-based progress reports to provide a more accurate, real-time picture of development. https://orca.cardiff.ac.uk/id/eprint/47510/1/2013AlyahyaSPhD.pdf

According to this thesis, such an approach can reduce release bottlenecks, support better forecasting, and increase situational awareness for distributed teams – by capturing not only human – reported status but also technical indicators automatically.
https://orca.cardiff.ac.uk/id/eprint/47510/1/2013AlyahyaSPhD.pdf

Furthermore, Kai Stapel et al.'s 2021 Flow Mapping: Planning and Managing Communication in Distributed Teams presents a method for methodically planning and controlling information flows in distributed projects. The writers suggest explicitly mapping communication flows and artifacts, tracking conformance to the communication strategy, and directly including information flow planning into project design.

Implication for Capstone: These findings suggest that a hybrid approach – combining manual reporting, technical signal tracking, and structured communication flow planning – improves progress tracking and visibility in outsourced/distributed projects. That aligns with plan to use tool – based automation (e.g., via Azure DevOps metadata, Teams bots) alongside formalized routines.

2.5 Role clarity, self – organization, and accountability in Agile/distributed teams

Agile methodologies often rely on team self – organization, shared ownership, and trust rather than command – and – control. However, distributed settings pose challenges to self – organization and accountability when communication is asynchronous and informal.

In Self-organizing Roles in Agile Globally Distributed Teams (2021), Sherlock A. Licorish & Stephen G. MacDonell study how roles and behaviors emerge in globally distributed agile teams. Their findings indicate that while self – organization is possible, in many teams certain individuals – often leads or senior programmers – become central coordination points, which can lead to informal overload, uneven accountability, and bottlenecks (Baiyere, Salmela, & Tapanainen, 2021).

This phenomenon resonates with context, where the architect ends up shouldering extra coordination and urgent task resolution due to lack of transparent status reporting. It demonstrates that without clear, formalized processes and role definitions, self – organization can lead to overload on a few individuals.

Implication for capstone: Structured reporting and automation help to formalize roles and responsibilities for status updates and distribute responsibility, so reducing reliance on informal self-organization and preventing overload on particular team members—e.g., the architect.

2.6 Emerging AI – assisted and tool – based reporting, automation and information flow

While the academic literature on artificial intelligence – assisted project management is still developing, tool-supported coordination and reporting is becoming increasingly common. Automated dashboards, reminders, integration between collaboration tools and task tracking systems are increasingly pushed to support transparency and status tracking in practice and industry – oriented literature. For tasks that become stale or un-updated, for instance, digital information radiators (dashboards) coupled with automatic notifications are said to "boost transparency and reduce manual reporting overhead." The Interaction Design Foundation

Moreover, using rule-based automation (alerts, reminders, triggers) becomes a realistic "lightweight AI – assisted" mechanism – without requiring ML models or extensive historical data as more distributed teams adopt cloud – based PM tools and integrate communication platforms (e.g., Azure DevOps + Microsoft Teams). Together with structured procedures, automation helps to institutionalize update systems and guarantee real-time visibility among many stakeholders.

Implication for capstone: The proposed Task Visibility and Predictability Framework – combining process formalization with AI-assisted automation – is not only theoretically justifiable, but also practically plausible and aligned with emerging industry practice.

2.7 Synthesis: key patterns & gaps

From the literature review, several recurring patterns emerge:

- **Formalization of processes and governance improves predictability and success** in project and portfolio management, especially under complexity. (Too & Weaver, 2014)
- **Visualization and information radiators** (digital or physical boards/dashboards) help agile teams externalize progress, improve transparency, and coordinate more effectively. (Sedlmair, Meyer, & Munzner, 2013).
- **Hybrid approaches to progress tracking,** combining manual updates and automatic signals (e.g., code commits, builds, test results), have proven advantageous in distributed agile contexts. (Alyahya, 2013).
- **In distributed and outsourced teams**, without clear roles and formalized communication flows, self – organization often leads to uneven workload and information bottlenecks. (Baiyere, Salmela, & Tapanainen, 2021).
- **Automation and tool – based reporting,** even without advanced ML, can reinforce formalization and maintain up – to – date visibility across teams.

At the same time, there remain gaps in both academic literature and practice:

- Few empirical studies examine **lightweight AI – assisted automation** (reminders, triggers, dashboards) combined with formalized task – status routines in small-to-medium outsourced IT projects.
- Most formalization research focuses on portfolio or multi – project level; less attention is paid to **task – level formalization and reporting** within a single project team.
- Many studies assume either **manual formal reporting or fully manual agile coordination** – not hybrid approaches combining automation with human discipline.
- There is limited guidance on how to adapt formalization and visibility frameworks for outsourced, distributed Agile teams under real – world constraints (time pressure, client dynamics, priority changes).

2.8. Conclusion – relevance to this capstone

Particularly in complicated, distributed software development environments, the literature powerfully supports the idea that **formalization, visibility, and tool-supported reporting** are essential enablers of predictable delivery. This capstone intends to design a **Task Visibility & Predictability Framework** combining process formalization and lightweight automation – is thus well-founded in theory and supported by empirical findings.

At the same time, the gaps identified – especially the lack of studies combining these elements at the task level in outsourced teams – highlight the originality and practical value of work. By integrating formalization, visualization, role clarity, and AI – assisted automation within an existing outsourced IT project, this research can contribute a meaningful, replicable model that addresses a real challenge many organizations face today.

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Research design (qualitative analysis)

Qualitative analysis is a research approach focused on understanding human behavior, experiences, communication patterns, and the meanings individuals assign to their actions and work processes. Unlike quantitative techniques, which depend on numerical data and statistical measurement, qualitative analysis investigates rich, detailed stories that expose how people view problems, make decisions, interact with others, and negotiate organizational structures. In the framework of this project, which aims to investigate why task completion predictability is low and how communication behaviors shape workflow visibility – qualitative methods are especially suitable. The problems seen in the outsourced IT project center on subjective interpretations, informal communication practices, unspoken norms, personal decision-making processes not easily quantified by numerical measurements alone. Furthermore mostly behavioral and context-dependent are the underlying reasons of irregular task updates, unclear ownership, delays, context switching, and miscommunication. Thus, by letting participants describe their difficulties and reasoning in their own words, qualitative analysis helps to provide better understanding of these fundamental elements.

I employed qualitative analysis utilizing an inductive open coding approach consistent with grounded theory principles to examine the interview data. Also with an AI – assisted tool QualiGPT (KindOPSTAR, 2023) to support first-cycle open coding. As input was provided the full dataset in CSV format and a fixed prompt specifying: a) inductive open coding; b) at least five codes per response; c) the next output structure (Question Number; Participant Role; Participant Name; Answer; Open Codes). QualiGPT produced an initial coded dataset, which I then reviewed and refined manually. Then merged redundant codes, adjusted wording, and created a final human – validated codebook that was applied consistently across the data. Thus, the AI acted as a coding assistant, while I remained the primary coder. To ensure reproducibility, I provide: (1) the original CSV, (2) the full coded CSV, (3) the exact prompt/instructions used, and (4) the final codebook. All quantitative summaries (code frequencies, distributions across questions, etc.) are computed using Python script, so the numerical results are fully reproducible.

The first cycle of study was open coding, in which every interview response was broken out into logical chunks and given brief conceptual labels. Three complementary coding methods were applied, guided by Open Coding in Qualitative Research (ResearchGate, 2025). Most often,

the core meaning of participant statements was summarized using **descriptive coding**—especially when participants described estimate behavior, communication preferences, or blockers escalation patterns. When participants reported actions, behaviors, or dynamic sequences inside the workflow—e.g., "context switching, "waiting on business decisions, "investigating anomalies"—**process coding** proved especially helpful for spotting trends in task handovers, communication, and delays. **In vivo coding** was applied to preserve the participants' own language, especially when terms such as "priority changes," "definition of done," "multitasking," or "forgotten tasks" emerged repeatedly across roles. These three coding techniques allowed to capture both what participants say and what they do, which is essential for analyzing workflow and behavioral disruptions in complex project settings.

Following open codes generated for every interview, the second – cycle **axial coding** step was used to group related codes into higher – order categories and **thematic analysis** uses for developing larger themes about task visibility, communication behaviors, workflow predictability, and organizational constraints; the final **interpretation** step – explaining patterns and underlying root causes that inform the design of the proposed process – formalization framework.

3.2 Interview participants selection

Participants were selected using a purposeful sampling strategy, targeting individuals whose daily responsibilities directly influence task completion, workflow visibility, and communication processes within project. Because the study's goal was to understand behavioral, procedural, and communication – related factors affecting predictability, only roles involved in requirement clarification, implementation, task updating, testing, technical oversight, and coordination were included. A total of 12 participants were selected, representing the full workflow ecosystem: 7 – Data Engineers, 2 – QA Engineers, 1 – Data Tech Lead, and 2 – Business Analysts. These individuals were specifically selected since they have practical knowledge of the task management techniques, channels of communication, and sprint execution dynamics of the project, so enabling them to offer special insights into the problems indicated in the problem statement. Every chosen position offers a necessary viewpoint on the difficulties investigated. With most of the delivery team, **Data engineers** handle tasks, daily basis status updates, ETAs, and blocking escalation; hence, their experiences expose fundamental trends behind delays, context switching, and inconsistent updates. Operating at the last phases of the

process, **QA Engineers** regularly find problems with incomplete handovers, unclear requirements, missing visibility, or early deployments; hence, their observations are crucial for comprehending downstream consequences of poor process alignment. Business analysts help to clarify requirements, acceptance criteria, quality of acceptance, and changes implemented mid-sprint key factors found in the problem statement as causes of uncertainty. Ultimately, the **Data Tech Lead** presents a cross-functional viewpoint on architectural dependencies, communication overload, and coordination problems resulting from changing priorities or blockers. These roles taken together offer a complete, end – to – end knowledge of the elements influencing task completion predictability and work visibility in IT projects.

3.3 Data collection via structured interviews

Data for this study were collected through structured individual interviews, designed specifically to examine the behavioral, communication, and workflow – related factors contributing to low task completion predictability and limited work visibility in the outsourced IT project. Interview questions presented in Appendix A. The interview protocol consisted of next sections (A–I), each targeting a distinct dimension of the problem identified in the project's problem statement – namely inconsistent task updates, unclear ownership, communication delays, unpredictable ETAs, dependency on overloaded roles, and lack of standardized processes. These sections ensured that interviews captured both operational practices and participants' subjective interpretations, providing a rich foundation for identifying systemic patterns and improvement opportunities.

Section A – "Understanding current task – update practices" investigated how team members now update task statuses, what causes status changes, and what usually is included. This part was crucial since one of the main causes of bad workflow visibility was found to be inconsistent or insufficient updates.

Section B – "Visibility of work progress" looked at whether participants thought the team, PM, or client could see their work. This resolved the fundamental issue of limited transparency, which directly affects coordination and accuracy of planning.

Section C – "ETA accuracy and task completion predictability" - what factors affect their accuracy, and how comfortable participants feel updating them. This part helped identify root causes including unclear criteria, task switching, and investigation time since erratic task completion was a major complaint.

Section D – "Blockers, interruptions, and communication flow" focused on toward escalation paths, blocker handling, and preferred communication channels. This helped one to grasp the causes of delays and the ways in which communication either speeds up or slows down advancement.

Section E – "Role clarity and coordination overload" covered who participants turn to for explanation and how often architects or leads step in. This part directly relates to the recorded problem of architect overload and its impact on team effectiveness.

Section F – "Perception of project management and sprint execution" asked participants how predictable sprints feel, what makes planning difficult, and how mid – sprint changes impact their work. This aligns with the problem of shifting priorities and unstable scope within sprints.

Section G – "Pain points and current inefficiencies" let participants express frustrations, workflow breakdowns, and particular instances of inefficiency, so fostering a better knowledge of systematic problems transcending mere surface level symptoms.

Section H – "Readiness for formalization and structured processes" asked participants whether they thought formalizing processes would either benefit or impede them. This is absolutely important since the Capstone suggests implementing organized tasks, revised guidelines, and clearer Definition of Done practices.

Section I – "Attitudes toward automation (AI – assisted communication)" explored whether participants would accept automated reminders, AI – driven nudges, and other automated support mechanisms. This section directly feeds into the project's second major solution direction: AI – assisted communication automation.

These sections guarantee that the data collecting process methodically addressed every aspect of the problem, from personal behavior and communication patterns to structural inefficiencies and readiness for artificial intelligence supported interventions. This method helped the research to ground the suggested recommendations in real-world team experiences and acquire a thorough knowledge of the elements restricting task predictability and work visibility.

Table 1. Mapping interview sections A–I to research questions and problem areas

Interview section	**What it examines**	**Research questions addressed**

Section A – Task – Update Practices	How tasks are updated, triggers, content of updates, frequency	RQ1: Factors contributing to inconsistent task updates and low visibility
Section B – Visibility of Work Progress	Whether work is visible to PM/team; gaps in perception; clarity of status	RQ1: Causes of low transparency; RQ2: Impact of limited visibility on coordination
Section C – ETA Accuracy & Task Predictability	How ETAs are estimated; comfort revising; reasons for delays	RQ3: Factors influencing ETA accuracy and task predictability
Section D – Blockers, Interruptions, Communication Flow	Types of interruptions; escalation paths; communication channels used	RQ1: Communication influence on visibility; RQ3: Blockers contributing to unpredictability
Section E – Role Clarity & Coordination Overload	Who participants ask for help; overload of architect; dependency patterns	RQ4: Impact of role ambiguity and bottlenecks on workflow and predictability
Section F – Perception of Project Management & Sprint Flow	Sprint predictability; planning challenges; mid – sprint changes; priority shifts	RQ3: Causes of sprint unpredictability; RQ5: Barriers hindering reliable delivery
Section G – Pain Points & Inefficiencies	Frustrations, inefficiencies, examples of delays or misalignment	RQ1–RQ5: Cross – cutting insights revealing root problems
Section H – Readiness for Formalization	Attitudes toward structured processes, task – update rules, Definition of Done	RQ6: Team receptiveness to workflow formalization
Section I – Attitudes Toward Automation (AI)	Openness to automated reminders, AI nudges, bots for updates	RQ7: How AI – assisted communication can improve predictability and reduce communication debt

Interview sections were designed to align directly with the study's research questions and the core problems identified in the project environment.
Sections A–C diagnose behavioral and process – related sources of unpredictability, Sections D–F reveal communication and coordination challenges.
Sections G–I explore pain points, readiness for change, and openness to automation.

3.4 Data analysis procedure

The collected data were analyzed using a systematic open – coding procedure grounded in qualitative methodology. This approach was chosen because the goal of the study was to deeply understand the social, behavioral, and procedural factors that shape task – update practices, communication patterns, blockers handling, ETA predictability, and coordination challenges in an outsourced IT project. Qualitative analysis enables the researcher to capture the nuances of team members' lived experiences – particularly the hidden operational dynamics that often do not surface in quantitative work tracking systems such as Azure DevOps.

Open coding was conducted manually with support from structured AI – assisted extraction tools to accelerate code generation while ensuring that interpretive decisions remained human – led. AI was used solely as a text – processing accelerator – not as an analytical agent – to help break long interview responses into smaller conceptual units. This aligns with the observation in the literature review that AI can support knowledge management by processing unstructured textual information while final interpretation remains under researcher control (Tenhunen, 2023)

The coding procedure unfolded in four steps:

1. Data preparation. All interview responses (12 participants across four roles: Business Analysts, Data Engineers, Data Tech Lead, and QA professionals) were cleaned, translated when necessary, and formatted into a consistent CSV structure.
2. First – cycle open coding. Each answer – 33 questions per participant – was coded line – by – line. A minimum of 3–5 conceptual codes per question were generated to ensure granularity and depth.
3. Code clustering and comparison. Codes were then examined across participants to identify overlaps, contradictions, or recurring patterns.

4. Development of the axial category. Higher level categories reflecting systemic project issues including inconsistent update routines, ambiguity of acceptance criteria, reliance on architects, low predictability of ETAs, coordination overload, and communication fragmentation were created from related codes. Raw data and the developing theoretical model of why task visibility and delivery predictability break down in outsourced IT projects are conceptual bridged by these axial categories.

This procedure enabled the discovery of cross -role patterns, such as shared frustration with unclear requirements, inconsistent task – update discipline, logistical blockers introduced by long daily flow cycles, and overloaded key experts whose bottlenecks propagate delays across the system.

By means of this rigorous coding process, the qualitative data were converted into organized analytical insights directly influencing the problem framing, root-cause identification, and design of suggested interventions including communication formalization, standardized update protocols, and artificial intelligence – assisted reminder mechanisms. Their theme development is grounded in these results.

3.5 Ethical considerations and data confidentiality

Following accepted ethical standards for qualitative research, this study focused especially on informed consent, participant anonymity, and confidentiality. It was imperative to make sure participants could answer honestly without regard to negative repercussions since the interviews covered internal communication, coordination techniques, and task-management strategies. Every participant was advised on the aim of the study, the voluntary character of involvement, and their right to refuse to answer any question. Participants were anonymized using initials and role categories (e.g., Data Engineer, QA Developer, Business Analyst), so ensuring that individual responses could not be linked to employees; no personally identifiable or proprietary information was gathered.

The research process was kept under data confidentiality all through. Interview materials were kept safely and never distributed outside. Only anonymized text was handled when AI tools supported translating, formatting, or initial coding, so exposing no sensitive organizational information. The researcher made all analytical decisions including those related to coding and theme development. Results were presented in aggregated form, with an eye toward systemic

trends rather than individual performance, so preventing managerial abuse and so strengthening the ethical integrity and credibility of the study.

3.6 Limitations of the methodology

While the qualitative approach used in this study shed light on the behaviors, communication patterns, and workflow issues influencing task predictability in outsourced IT projects, several limits have to be admitted. First, the study included a rather small sample of twelve participants; although this group comprised important project roles—including Business Analyzes, Data Engineers, a Data Tech Lead, and QA Engineers—the results cannot be applied generally across all outsourced IT teams or more general organizational settings. The interviews took place inside the framework of one project, thus the behaviors, communication styles, and cultural standards found might not apply to other teams, project or business.

The self-reported character of the interview data presents another restriction. Recall restrictions, social desirability, or personal interpretations of workflow problems could cause participants to unwittingly offer biassed or incomplete accounts. While methodical open and axial coding aimed to reduce subjectivity, all qualitative research involves researcher interpretation by nature. Although the study used multiple open codes (in vivo, descriptive, and process) and transparent coding techniques to support analytic rigor, the subjective element cannot be totally eliminated. The practical reality of the project environment—that the researcher lacked access (due to NDA limitations) to operational system logs, historical task – update timelines, or quantitative metrics from Azure DevOps—results in another methodological limitation: the analysis cannot incorporate mixed-method triangulation. Consequently, the study stresses participants' points of view instead of contrasting self-reported behavior with real task-update data.

Ultimately, even if artificial intelligence (AI) helped tools sped up text processing and coding, they were not used for interpretation or theme generating. Although this guaranteed methodological integrity, hand verification of all codes naturally limited the scope of qualitative data available for analysis. AI helped to improve efficiency by supporting the breakdown of interview material; it did not completely eliminate possible human supervision in assigning the most representative codes or interpreting complex answers. Notwithstanding these constraints, the approach used in this Capstone provides a strong and relevant basis for comprehending the systematic causes of erratic task completion and limited visibility in outsourced IT environments.

CHAPTER 4 – FINDINGS & ANALYSIS

4.1 Overview of participants

Twelve interview subjects for the study played important roles on the outsourced IT delivery team. These roles were purposefully chosen to reflect a thorough, multi-perspective knowledge of how task updates, communication patterns, blockers handling, estimation behaviors, and workflow dependencies affect job completion predictability and work visibility. Including participants whose daily responsibilities span business analysis, backend data development, quality assurance, and technical leadership was crucial since the challenges of the project show on several layers of the delivery process.

These players combined to produce a representative and balanced dataset reflecting both operational and strategic points of view inside the project. Their combined experiences allowed the study to track how information flows (or fails to move) across roles, how responsibilities are understood differently, and how daily operations lead to systematic inefficiencies. Finding axial themes – such as inconsistent update routines, communication overload, unclear acceptance criteria, role overload, and coordination bottlenecks – that underlie the fundamental difficulties of the project depended on this diversity of points of view. The study guaranteed enough depth and breadth of insight by including voices from all team roles, so enabling a grounded knowledge of the root causes of limited work visibility and poor task predictability in IT project.

4.2 Presentation of key themes (from axial coding)

The main themes found from the axial coding phase of the qualitative data analysis are discussed in this part. Inspired by open coding and code improvement, related ideas were methodically arranged to find higher order trends explaining recurrent difficulties in task execution, communication, and predictability within the investigated outsourced IT project. The axial coding process concentrated on creating links between categories and tying specific actions to more general systemic problems influencing delivery performance.

Six main themes that together explain how low task completion predictability and decreased work visibility are caused by gaps in structured communication, process formalization, and coordination mechanisms were found by the analysis. These themes span roles (Data Engineers, QA experts, Business Analysts, and Technical Leads), suggesting that the found problems are not role-specific but rather systemic in character.

Theme 1: Inconsistent task update behavior

This theme shows that Developers and QA update tasks in Azure DevOps in very different ways and at different frequencies – some in real time, some once per day, some only when reminded or before stand – ups. This directly belongs to STATUS_UPDATES_AND_VISIBILITY, because the board is not a reliable reflection of reality, forcing PMs and the architect to guess who is free or blocked. It also touches PROCESS_AND_FORMALIZATION: there are no shared rules about what "good" updating looks like (e.g., when to move to In Progress, when to add comments, how to revise ETAs). Skipped or late updates are often justified by WORKLOAD_PRESSURE_AND_TIME ("too many meetings", "no time to write comments"), and that weakens SPRINT_PLANNING_AND_PREDICTABILITY, since forecasts and capacity planning rely on outdated statuses. My ideas about standardized reporting and a Teams bot fall under TOOLING_AND_AUTOMATION as potential mechanisms to stabilize this behavior and reduce the gap between actual work and what the tools show.

Theme 2: Unclear or changing requirements

This theme combines ambiguous acceptance criteria, late clarifications from stakeholders, and frequent mid – sprint changes to user stories and bugs. It clearly sits in CLIENT_AND_SCOPE_MANAGEMENT, because many of these shifts originate from the client ("urgent" requests, out – of – scope changes, last – minute updates). That instability flows into ESTIMATION_AND_ETA, as people repeatedly mention that unclear ACs, hidden side effects and missing details make accurate ETAs almost impossible. It also affects QUALITY_AND_TESTING: unclear expectations, missing dev testing and late requirement changes lead to rework, retesting and production risk. At the planning level, this directly harms SPRINT_PLANNING_AND_PREDICTABILITY, because even a well – defined sprint backlog becomes unstable once requirements move during execution. Finally, the theme reveals missing PROCESS_AND_FORMALIZATION: instead of formal change requests and clear versioned requirements, changes often happen informally inside existing tickets, exactly as described in my project problem.

Theme 3: Communication overload and lack of standardization

Here, participants describe heavy use of COMMUNICATION_CHANNELS (Teams chats, calls, two daily stand – ups, ad – hoc pings), but without clear rules on what should go

where. Requirements clarifications, decisions, ETAs and blockers are scattered across chat, calls and ADO comments. This reflects a weak PROCESS_AND_FORMALIZATION of communication: there is no shared "contract" such as "all decisions and final clarifications must be in the story," so important context gets lost or duplicated. The sheer volume of calls and messages drives WORKLOAD_PRESSURE_AND_TIME concerns ("two hours of meetings per day", difficulty to re-focus afterward). My proposed solutions (structured updates, bots, automatic reminders) fall under TOOLING_AND_AUTOMATION, but participants warn that tools must reduce noise, not increase it. Finally, this theme mildly touches FEEDBACK_AND_RECOGNITION: people feel they provide status and reports, but are not always sure this information is noticed or valued, which contributes to frustration and demotivation.

Theme 4: Architectural and leadership bottlenecks

This theme shows that the architect, tech lead, and sometimes PM are central hubs for almost every significant decision: clarifications, design choices, cross – team issues, blocker resolution. That is clearly ROLE_AND_RESPONSIBILITY: responsibilities are concentrated rather than distributed, so the system depends heavily on a few key people. It also reflects BLOCKERS_AND_ESCALATION patterns – most blockers are escalated to the same individuals, who are simultaneously busy with client calls, incident investigations and coordination work. This leads to WORKLOAD_PRESSURE_AND_TIME for these roles (burnout risk, overtime, long queues for answers) and creates bottlenecks that delay delivery, which my original problem statement already highlighted. Because escalation paths are informal and person – based, rather than process – based, it also reveals gaps in PROCESS_AND_FORMALIZATION: there are no lightweight rules that would allow routine questions to be solved via standardized updates or shared documentation. Finally, there is an implicit FEEDBACK_AND_RECOGNITION element: developers often feel they are "chasing" decisions and not always getting timely, constructive feedback from leadership, which affects trust and perceived fairness.

Theme 5: Low task predictability due to interruptions and context switching

This theme integrates constant INTERRUPTIONS_AND_CONTEXT_SWITCHING: urgent client requests, sudden bugs, cross – team issues and priority changes that force developers to stop one task and jump to another. Tasks remain "In Progress" but are actually on

hold, which makes the board misleading. This dynamic attacks SPRINT_PLANNING_AND_PREDICTABILITY directly – ETA accuracy drops, sprint goals are missed, and people describe completion as fundamentally hard to forecast. Because estimates are created under one set of assumptions and then disrupted by dozens of small interruptions, ESTIMATION_AND_ETA becomes more about guessing than planning. Many of these interruptions come from CLIENT_AND_SCOPE_MANAGEMENT issues: out – of – scope work, mid – sprint "urgent" demands, and micro – management from the client. All of this increases WORKLOAD_PRESSURE_AND_TIME: overtime, stress, and a feeling of never catching up. This theme is a very strong bridge back to my research problem: team members confirm that the current ad – hoc handling of "urgent" work undermines both predictability and psychological safety.

Theme 6: Workflow visibility gaps

This theme focuses on the blind spots in how work is seen across teams and roles. Even with ADO boards and stand – ups, there are STATUS_UPDATES_AND_VISIBILITY gaps: invisible investigation time, cross – team dependencies (like deleted columns used by another team), tasks stuck in the wrong status, and parallel work causing merge conflicts. Because PMs, architects and the client do not see full context, SPRINT_PLANNING_AND_PREDICTABILITY suffers: planning does not reflect real effort, delays look like "lack of organization" rather than structural issues, and capacity decisions become guesswork. From a CLIENT_AND_SCOPE_MANAGEMENT viewpoint, these visibility gaps feed client dissatisfaction – my client perceives the vendor as unreliable precisely because they cannot see the true state of work and blockers. This theme is also linked to TOOLING_AND_AUTOMATION: people mention potential value in stale – task notifications, better progress indicators, and structured dashboards, as long as they reduce manual reporting effort. Finally, some visibility gaps directly affect QUALITY_AND_TESTING (e.g., deploying to UAT without proper QA sign – off, hidden dependencies in data structures), reinforcing the concern that rushing or skipping process steps degrades product quality.

4.3 Linking findings to research questions

This chapter interprets the qualitative findings obtained through open and axial coding and aligns them directly with the research questions defined in Chapter 1. This section should answer the

following main question “How do the empirical findings from the interviews directly answer the research questions of the study?”. This chapter moves from “what was found” → “what it means in relation to the research questions.”

Findings related to RQ1.

“How does the lack of formalized communication and status – update routines impact task completion predictability in outsourced IT projects?”

1. Relevant themes

- Theme 1: Inconsistent task update behavior
- Theme 5: Low task predictability due to interruptions and context switching
- Theme 6: Workflow visibility gaps”

2. How these themes answer the question

Taken together, these ideas highlight how personal habits, urgency, or outside pressure rather than consistent rules drives task updates **without formal routines.** Rather than at consistent checkpoints, developers and QA often update tasks at the beginning and end of work – or only when a demo, release, or urgent question arises. This leads to situation when **status fields and ETAs lag behind reality**, so even when work is progressing, the board does not reliably show it.

Theme 5 adds that constant interruptions, urgent bug fixes, and task switching break the link between planned work and actual execution. When people switch tasks, they frequently forget to move items back to “New”, so tasks remain “In Progress” while receiving no attention. Theme 6 shows that this behavior creates structural visibility gaps: long – running tasks with no comments, invisible investigation work, and blocked tasks that look active on the board. The combination of irregular updating + heavy context switching makes it very difficult to predict when tasks will actually finish, because the ADO does not show the true state of the work in a timely or consistent way.

3.Patterns across participants and roles: Data Engineers described forgetting to change states when switching to more urgent tasks, leaving items in “In Progress” even when paused. They also mentioned that ETAs are hard to update because requirements are unclear and context changes mid – sprint.

- QA roles (YS, DB) pointed out invisible effort—investigation, coordination, proving problems—that isn't shown in task updates and observed that delays in updates also affect implementation and testing. They clearly link late updates to decreased output.
- Business Analysts (TK, SS) reported that their own tasks tend to be up – to – date, but acknowledged that delays caused by stakeholders or other teams are often not fully visible in the board until late.
- Tech lead confirmed that some tickets sit in "In Progress" for longer than they should and that non – priority items can "hang" in one status for a long time. They see that predictability is heavily affected by this behavior.

People know that the board is not a real-time reflection of work, and they compensate across roles using stand-ups and ad hoc communication – at the expense of predictability.

Findings related to RQ2

"What process – formalization mechanisms can be introduced to ensure consistent, accurate, and timely task – status updates in Azure DevOps (ADO)?"

1. Relevant themes:

- Theme 1: Inconsistent task update behavior
- Theme 2: Unclear or changing requirements
- Theme 3: Communication overload and lack of standardization
- Theme 6: Workflow visibility gaps

2. How these themes answer the question

The themes collectively point to **specific formalization mechanisms** that participants implicitly ask for:

- From Theme 1, we can infer the need for **clear rules on when and how to update tasks**: e.g., at task start, after meaningful progress, when blocked, when requirements change, and when ETAs shift. Many respondents already follow their own micro – routines (e.g., daily updates, beginning – and – end only, or after PR merge), but these are inconsistent and not shared.
- Theme 2 shows that **unclear and shifting requirements** undermine the usefulness of any update. Participants explicitly mention unclear/late acceptance criteria and changing

business expectations. This suggests formalization of **requirements templates, AC structure, and change – handling rules,** so that status and ETA updates are grounded in a stable understanding of "done".

- Theme 3 highlights that communication is high – volume but **not standardized by channel or format**. Formalization mechanisms here would include conventions such as: *"All implementation/testing notes and blockers must be documented in ADO comments; Teams is for synchronous clarification only."* This would reduce duplication and ensure that ADO becomes the single source of truth.
- Theme 6 indicates the need for formal rules about what **must be visible in ADO**: e.g., *investigation work, dependency information, evidence (screenshots, queries) and explicit "on hold" states*. Formalization would require minimum content in ADO before a task can move to certain statuses ("Ready for QA", "Closed", etc.).

Therefore, the data implicitly define a **process blueprint** – status triggers, minimal fields and comments, requirement templates, and channel conventions – that could be codified as formal mechanisms, so transcending mere complaining.

3. Patterns across participants and roles

- Data Engineers and QA consistently refer to "good practice" elements: adding screenshots, logging queries, tagging blockers, documenting root cause, and leaving comments for the next person. The problem is that these practices are **personal and uneven,** not enforced or shared as rules.
- BA roles emphasize the need for **clear AC and better requirement descriptions,** and suggest the use of **task trackers / visual progress indicators** to make progress more explicit.
- Tech lead and QA explicitly mention the idea of documenting a clear process on a WIKI and having "definition of done" clarified per role. They see the gap in shared understanding and view documentation as a key formalization tool.

Although there is general agreement across roles that formalizing "when, what, and where" to update – together with structured requirements – is both needed and acceptable as long as it does not consume too much development time.

Findings related to RQ3

“How can AI – assisted communication automation (such as Teams reminders, rule – based triggers, and stale – task detection) enhance real – time work visibility?”

1. Relevant themes:

- Theme 1: Inconsistent task update behavior
- Theme 3: Communication overload and lack of standardization
- Theme 6: Workflow visibility gaps

2. How these themes answer the question

These themes show that AI – assisted automation is most valuable where human routines are fragile and manual chasing is expensive:

- Theme 1 demonstrates that people forget updates, postpone them, or rely on memory. Here, AI can **provide reminders for stale tasks, overdue ETAs, or missing status changes** (e.g., items “In Progress” with no update for N days).
- Theme 6 identifies specific visibility gaps – long – running tasks with no comments, paused work not marked as paused, investigation effort not logged, and invisible blockers. Rule – based triggers and stale – task detection **could surface these anomalies automatically, prompting the assignee** (or their lead) to add necessary context.
- Theme 3 indicates that PMs and architects currently have to **chase information via standups and chats**. Automation can shift this burden: instead of humans checking every item, the system can proactively flag problematic patterns, send short prompts via Teams, and ensure information flows back into ADO rather than dispersing across chat threads.

Basically, the themes imply that by pushing timely updates, stressing differences between board and reality, and lowering manual coordination effort required to maintain ADO accuracy, AI – assisted automation improves real-time visibility.

3. Patterns across participants and roles

- **Many participants** express openness to **automated reminders**, especially for stale tasks and blockers, provided they are not too frequent or distracting. They see reminders as helpful for memory support rather than as surveillance.
- **Some engineers** are more skeptical, indicating that reminders alone do not solve deeper process and requirement problems. They stress that automation only makes sense if aligned with meaningful rules, not just “more notifications.”

- **QA and BA roles** tend to appreciate anything that reduces time spent in status meetings and manual follow – ups, and are more positive about bots prompting short, structured updates rather than long calls.

Overall, there is a pattern of conditional acceptance: AI automation is welcomed when it is **lightweight, tied to clear rules, and reduces manual overhead**, but rejected if it simply adds noise.

Findings related to RQ4

"In what ways does improved transparency reduce coordination overload on key roles such (project architect, etc.)?"

1. Relevant themes

- Theme 3: Communication overload and lack of standardization
- Theme 4: Architectural and Leadership Bottlenecks
- Theme 6: Workflow visibility gaps

2. How these themes answer the question

The themes show that poor transparency forces architect and leads into a **constant coordination role.** Because ADO does not reliably show current status, and communication channels are used inconsistently, architects become the "human integration layer":

- Theme 4 shows that architect and tech lead are **frequently consulted for clarifications, escalations, and cross-team issues**, and that they spend a lot of time in meetings and Q&A sessions. Much of this work is **compensating for missing or unclear information** in tasks and requirements.
- Theme 3 indicates that status and decisions are distributed across chats, calls, and personal notes, rather than converging into a shared artefact. That forces key roles to constantly **rebuild the system-wide picture** by talking to multiple people.
- Theme 6 demonstrates that visibility gaps – such as stale tasks, missing context, and invisible blockers – mean that architects cannot trust the board to make decisions, so they must **manually probe**, ask "What is going on with this ticket?", and break down misunderstandings in meetings.

Through standardized updates, improved ADO documentation, and artificial intelligence supported completeness checks, improved transparency would enable many daily clarifications

back into the task system**.** Answers for team members might come from the ticket instead of the architect. This helps to lessen coordination overload: the architect would still handle difficult design and cross-team decisions, but she would not have to obsess over daily visibility.

3. Patterns across participants and roles

- **Participants** (OP, VM, DB, TK) consistently describe architect as overloaded, attending numerous meetings, and being the main person others go to when they don't know who else can answer a question.
- **Developers and QA** often say they go to the architect or team lead when requirements are unclear or when they do not know who owns a problem, reinforcing the bottleneck pattern.
- Some participants explicitly note that **if stories and tasks contained better descriptions, impact analysis, and sign – off rules**, fewer questions would need to go to the architect.

Across roles, the shared perception is that **improved transparency in ADO would allow more self-service**, reduce the need for repeated "status checks" with the architect, and limit their involvement to genuinely architectural decisions.

Findings related to RQ5

"What measurable improvements – such as reduced delays, fewer escalations, or more accurate ETAs – can be achieved after implementing a formalized and automated task – visibility framework?"

1. Relevant themes

- Theme 1: Inconsistent task update behavior
- Theme 2: Unclear or changing requirements
- Theme 3: Communication overload and lack of standardization
- Theme 5: Low task predictability due to interruptions and context switching
- Theme 6: Workflow visibility gaps

2. How these themes answer the question

Because this study is qualitative, the themes don't provide numeric improvements but indicate the dimensions where improvements should appear once the framework is implemented:

- Theme 1 and Theme 6 suggest that clear rules + automation will lead to **fewer stale tasks,** fewer tickets stuck in the wrong status, and **more up-to-date ETAs and**

comments. This can be translated into measurable indicators such as: number of tasks without update in N days, percentage of tasks with ETA vs actual completion close to each other, etc.

- Theme 2 implies that with standardized requirements and clearer AC, ETAs will be **more accurate,** and rework due to misunderstood scope should decline. This can be tracked through metrics like re-opened tasks or number of ETA revisions.
- Theme 3 indicates that standardized communication and automation should reduce **manual escalations and status-chasing**. We can measure this via frequencies of "status – only" pings or the amount of time spent in status – focused meetings.
- Theme 5 suggests that by formally handling interruptions and context switching (e.g., marking tasks as paused/on – hold and surfacing unplanned work), we can reduce hidden delays and better account for unplanned disruptions in planning. This supports improved **predictability across sprints.**

The themes point to measurable improvements in: **timeliness of updates, ETA accuracy, number of stalled or stale tasks, frequency of escalations, and alignment between planned and actual completion.**

3. Patterns across participants and roles

    - Most roles believe current state is suboptimal but improvable, and many explicitly mention where they expect gains: fewer surprises at demos, less chasing in meetings, fewer merge conflicts or rework due to missed communication.
    - QA and leads emphasize that better dev – testing, clearer requirements, and more information in tasks would directly reduce rework and delays in testing.
    - Some engineers and analysts highlight that predictability will never be perfect (because of client behavior, changing scope, or system complexity), but they still expect relative improvements if updates and communication become more disciplined and supported by automation.

Across participants, there is a shared expectation that formalization + automation will not make the project perfectly predictable, but will reduce avoidable delays and visibility failures, which can then be monitored via simple, project – level metrics.

4.4 Interpretation and implications

This part analyzes the empirical results and theme - RQ analysis (Chapter 4.3) in respect to the central problem of this Capstone: the lack of a formalized, transparent, and consistent process for reporting task progress and work schedules in an outsourced IT project. The debate also relates to the goals of the research and highlights useful consequences for the design and acceptance of the suggested Task Visibility and Predictability Framework.

From Individual Habits to Systemic Failure Modes

One of the most important insights from the data is that **the visibility and predictability problems are systemic, not individual**. Across roles, participants describe themselves as conscientious: they update tasks "once a day," "at the beginning and end," or "whenever new information appears." Many add detailed comments, screenshots, queries, or blocker tags. Yet despite this effort, the project still suffers from **unpredictable completion, frequent surprises, and overloaded architects and PMs**.

The themes **Inconsistent task update behavior** and **Workflow visibility gaps** show how this paradox emerges. Updates are driven primarily by **personal habit, urgency, or local judgment**, rather than by shared rules. For example, some developers treat a story as the main documentation artefact and keep tasks minimal, while others document every investigation step in the task. Some update status only at start/end, others after every meaningful piece of progress.

At the same time, theme **Low task predictability due to interruptions and context switching** reveals that the work system is inherently volatile. Priorities change within the day; unplanned bugs and "urgent client wish-lists" appear mid-sprint; people switch to other tasks without always updating states. The result is not just delayed tasks, but **misleading signals**: items remain "In Progress" but are effectively on hold; investigation work is performed but not visible; blockers are known within the team but not encoded in the tool.

From management perspective, this means the root issue is not that team members are careless or unmotivated. Instead, the **combination of high variability in work and a lack of codified routines** makes it impossible for individual "good intentions" to scale into predictable, project-level behavior. This directly supports **Research Objective 1** (analyzing root causes): the findings make clear that **unformalized, heterogeneous status-update practices** and **unmanaged context switching** are structural drivers of unpredictability.

Process formalization as a coordination mechanism, not bureaucracy

The themes strongly suggest that **formalization is needed**, but they also highlight how it should be designed to avoid becoming yet another burden. The theme **Unclear or changing requirements** shows that many ETA and status problems originate before implementation even starts. Developers and QA repeatedly refer to **ambiguous acceptance criteria, late updates to requirements, and incomplete business context**. Under these conditions, even the most disciplined status updates become unstable: ETAs shift, tasks are re-scoped, or reopened. This implies that process formalization cannot be limited to "update more often"; it must also address **the quality and stability of the input**, especially AC and story descriptions.

In parallel, theme **Communication overload and lack of standardization** highlights that communication volume is already high: two daily stand-ups, multiple Teams chats, ad-hoc calls, and frequent cross-team clarifications. The problem is not a lack of communication, but that **channels and formats are not standardized**. Teams is often used as the primary source of truth, with key decisions and clarifications buried in chat history instead of recorded in Azure DevOps. This produces duplication and forces PMs and architects to reconstruct context from multiple sources.

The data therefore implies a specific interpretation of **RQ 2** (process-formalization mechanisms):

- Formalization should define **clear triggers and minimal content** for updates (e.g., when starting work, when blocked, after requirement change, when switching tasks, when ETA changes).
- Stories and tasks require **standardized templates** for acceptance criteria, impact analysis, and test conditions, so that progress and ETA relate to a stable notion of "done".
- Communication rules should explicitly state that **Azure DevOps is the canonical source of status and implementation details**, while Teams is a synchronous support channel, not a substitute for task documentation.

Importantly, several participants explicitly state that formalization is acceptable **if it does not steal time from core work**. This is a critical managerial implication: **formalization must be "lightweight but explicit"** – clear enough to align behavior, but efficient enough that developers and QA can comply without feeling policed or overburdened. This directly aligns with **Research Objective 2**, which aims to design clear yet pragmatic reporting rules.

Rethinking roles: from heroic architects to distributed responsibility

The theme **Architectural and leadership bottlenecks** illustrates a classic anti-pattern in distributed engineering projects: when tools and processes do not provide dependable visibility, **leaders are forced to become the manual glue**. Architects and leads are consulted for clarifications, blockers, cross-team coordination, and sometimes even basic status understanding. Many participants describe the architect as constantly present in meetings, groomings, Q&A sessions, and chat threads.

This pattern has two implications:

1. **Leadership time is diverted from high-leverage architectural work** to low-level coordination and status chasing.
2. **Team members implicitly learn that "real" answers reside with the architect**, not in the stories or tasks. This further reinforces the dependency loop.

According to **RQ4**, the study asks how improved transparency could reduce this overload. The findings suggest that if **stories, tasks, and ADO comments were consistently richer and more self-contained**, many of the questions that currently go to the architect could be answered directly from the board. Several participants explicitly describe their ideal state as "all necessary information in the ticket" and call for a clear process description on a wiki page.

For Engineering Management, this points to a **shift in role design**:

- Architects and leads should be responsible for **defining and modeling the process and standards**, not for repeatedly compensating for their absence.
- Ownership of reporting must be **distributed across roles**, with each story and task having clear responsibilities for updates, AC quality, and blocker documentation.
- Leadership behaviors (e.g., consistently using the board as the primary reference, refusing to make decisions based on undocumented chat messages) become **a key enforcement mechanism** for process formalization.

This directly supports **Research Objective 4**, which focuses on reducing architect overload by improving transparency rather than by "working harder."

AI-assisted automation as a reinforcement layer, not a replacement

The themes related to automation clearly show that **AI-assisted mechanisms are seen as helpful, but only under certain conditions**.

Participants generally welcome **reminders for stale tasks, blockers, or sprint-end clean-up**, especially when they reduce the need for long status calls. However, more skeptical voices emphasize that notifications alone cannot fix unclear requirements, unstable scope, or poorly defined roles. The theme **Communication overload (two standup meetings per day) and lack of standardization** also warns that if bots simply generate more noise on top of existing chats, they may worsen the problem. Viewed through **RQ 3**, the role of AI-assisted automation is best interpreted as:

- **Nudging compliance** with the formalized process: asking for updates when tasks appear stale, ETAs expire, or status is inconsistent with recent activity.
- **Highlighting anomalies**: surfacing tickets that are "In Progress" without updates for N days, or tasks near demo dates with no recent comments.
- **Reducing manual coordination effort**: instead of architect manually asking about potential issues, the system can produce a prioritized list of items that need attention.

This aligns directly with **Research Objective 3**, which aims to implement lightweight AI-assisted mechanisms using existing tools (Teams, Azure DevOps). The empirical data imply that such automation must be:

- **Configurable and low-frequency**, to avoid notification fatigue.
- Tightly coupled to **clearly understood rules** (e.g., "In Progress + no comment in N days"), so users see the logic and value behind each prompt.
- Oriented toward **support rather than surveillance**, maintaining psychological safety and trust.

For an Management audience, the implication is that AI should be introduced **as part of a socio-technical system** – complementing process formalization and leadership practices, rather than acting as a purely technical fix.

Implications for sprint planning, metrics, and continuous improvement

Finally, the themes **Low task predictability due to interruptions and context switching** and **Unclear or changing requirements** show that sprint predictability problems are both **structural and behavioral**. Participants repeatedly note very short sprints, late requirement changes, and frequent urgent requests. They also acknowledge that some unpredictability is inherent given the project's scale and client behavior.

However, the data also indicate that **a significant portion of unpredictability is avoidable**. Tasks are sometimes re-opened because critical information was missing from the story. ETAs are optimistic because unplanned coordination and investigation time is not fully considered. Context switching is not reflected in status and planning.

This informs **RQ 5**, which focuses on measurable improvements after implementing the framework. While this Capstone does not conduct large-scale quantitative evaluation (by design), the qualitative results point to **specific metrics that Managers can track in future iterations**, such as:

- Number or percentage of **stale tasks** (no update for N days).
- Ratio of tasks with **ETA close to actual completion** vs significantly late.
- Frequency of **status-only escalations** to architect/PM (e.g., "what is going on with this ticket?").
- Number of **reopened tasks** due to unclear requirements or missing information.
- Time spent in **status-focused meetings** before and after introducing automated reminders and clearer rules.

These measures fully complement the goal of the research and study by operationalizing the expected advantages of the Task Visibility and Predictability Framework and offering a feedback loop for ongoing development.

Summary: implications for project management practice

The interpretation of themes across all research questions reinforces the central problem statement: **the lack of a formalized, transparent, and consistent process for reporting task progress leads to unpredictable completion, poor visibility, and leadership overload**, even in a team of skilled and well-intentioned professionals.

The implications for Engineering Management are:

- **Treat visibility and predictability as design problems**, not as individual performance issues. The behavior observed is a rational response to an under-specified process and overloaded communication environment.
- **Introduce process formalization as a coordination mechanism**, built around clear rules, templates, and roles, rather than as heavy bureaucracy.

- **Use AI-assisted automation as a reinforcing layer** that supports human routines, highlights anomalies, and reduces manual overhead, not as a substitute for clear expectations.
- **Redefine leadership roles** so that architect and PM invest their time in defining standards, modeling behavior, and interpreting metrics, instead of manually gathering the project status from fragmented conversations.
- **Adopt simple, project-level metrics** that make the impact of the "Task Visibility and Predictability Framework observable and discussable, enabling iterative refinement rather than a one-time "process rollout."

Taken together, these implications show that the suggested framework directly addresses the goals of the Capstone and provides a pragmatic, socio-technical pathway for enhancing task visibility and predictability in outsourced IT projects similar to the studied here.

CHAPTER 5 – PROPOSED TASK VISIBILITY AND PREDICTABILITY FRAMEWORK

This chapter aims to translate the qualitative results of Chapter 4 into a methodical, actionable framework. The framework translates noted difficulties – such as uneven task updates, unclear requirements, communication overload, and architect overload—into a methodical, practical fix.

The proposed framework integrates **process formalization** with **AI-assisted communication automation**, leveraging existing enterprise tools (Azure DevOps and Microsoft Teams) to minimize overhead while improving transparency.

5.1 Process formalization recommendations

Addresses RQ2: Findings from Chapter 4 show that task updates are currently driven by **individual habits, urgency, or informal norms**, rather than shared expectations. While many team members act responsibly, the lack of explicit rules results in fragmented visibility and unreliable status signals for project management and leadership.

The goal of process formalization is **not to increase reporting time**, but to create **predictable, minimal, and role-appropriate routines** that ensure Azure DevOps to show the actual state of work. Based on the findings, the following formalization are recommended:

Standardized update rules. The framework proposes the following **mandatory update triggers**, applicable across roles:

1. **Task starts**
    - User Story moved to *In Dev*
    - Task Status must be moved from *New* to *In Progress*
    - A short comment confirming task start and any message in case any risk
2. **Blocker encountered**
    - Status left in *Active*, Added Tag = '*Blocked'*
    - Comment describing:
        - Must have comment with blocker type (technical, requirement, environment)
        - If external dependency exists (name of external person)
        - Expected next action

3. **Context switch or task pause**
    - Into current task that in *In Progress* state need to add comment indicating:
        - Switched to task number
        - Reason for pause (priority change, urgent task, dependency wait)
        - Whether the task is actively blocked or temporarily de-prioritized
4. **ETA Change**
    - Mandatory comment explaining:
        - Why the ETA changed
        - What assumption was invalidated (requirements, dependencies, scope) that lead to ETA change
5. **Task Completion**
    - Task status moved to *Close.*
    - In task updated the *Efforts (Completed), Efforts (Remaining) set to 0.*
    - Added text and screen evidences into Task
    - Added comment into User Story with PR link, screenshots, queries, etc.

Role-based responsibilities. To avoid ambiguity, the framework defines **explicit reporting responsibilities**:

- **Developers (Data engineers, etc.)**
    - Maintain task and user story status accuracy
    - Confirm understanding of the User Story
    - Document blockers, investigation results with evidence
- **QA Engineers**
    - Update verification progress and discovered Bugs
    - Attach test evidence and clarify readiness for release
- **Business Analysts**
    - Ensure requirement changes and clarifications are reflected in stories
    - Document scope changes and acceptance-criteria (AC) updates with date when it is happened
- **Project Manager**
    - Monitor compliance with update rules
    - Use task data (not chat) for allocation and risk assessment

- **Architect**
    - Define and maintain tech standards for development process
    - Develop Tech documentation
    - Intervene only when Cross-task dependencies exist

This structure directly responds to findings related to **unclear ownership and leadership overload**.

5.2 AI-assisted communication solutions

Addresses RQ3: **Purpose of Automation.** Empirical results unequivocally show that human discipline by itself cannot ensure constant visibility under actual project conditions. Task switching, interruptions, and workload pressure cause forgotten updates and delayed context sharing. Automation is presented as a **reinforcement layer** supporting disciplined behavior with minimum cognitive overhead rather than as a control mechanism.

The proposed AI-assisted mechanisms include:

1. **Stale task detection** with
    - automatically mark (add Tag) the tasks that:
        - Remain *In Progress* without updates >= N days
        - Have passed ETA without comment
    - sends a **reminder (private or group chat)** in Teams to the task owner
2. **Rule-based nudges** with gentle prompts when:
        - A task is moved to *In Progress* without a comment
        - A blocker status (or Tag) is set without explanation
        - A task is closed by Developer or QA engineer without evidence attached
3. **Everyday reminder** with automated reminder before daily standup:
        - Update task status
        - Log remaining time
        - Confirm previous ETA or carryover with reason why
4. **Sprint-End reminders with** automated reminder before sprint end to:
        - Update task status
        - Log remaining time
        - Confirm readiness or carryover
5. **Exception highlighting** with aggregated alerts to PM/Architect for:

- Multiple blocked tasks
- High-risk items near demo or release dates

To avoid fatigue notification, automation follows these principles:

- Low frequency, high relevance
- Clear explanation of *why* a reminder was sent
- Focus on anomalies, not routine behavior
- Supportive tone (nudging, not policing)

5.3 Improving requirements clarity and definition of done (DoD)

Addresses RQ2 and supports RQ5: Findings consistently show that **unclear or changing requirements** are a major driver of ETA change and rework. To address this, the framework introduces a **standard AC template** for all stories:

- User story structure template
- User- and business-focused story writing
- Technical details in role-specific tasks
- Clarification through active grooming Q&A sessions
- Clear and unambiguous acceptance criteria (AC)

User stories cannot be moved into active development unless minimum above AC completeness criteria are met.

The framework also defines a **role-specific DoD**, clarifying what "done" means for each role:

- **Development DoD:** code merged, unit/dev testing with evidences completed, technical notes added
- **QA DoD:** test cases created and executed, evidence attached, critical bugs resolved or documented
- **Business Analyst DoD:** requirements validated, AC finalized, User Stories approved with stakeholder

When requirements change in the middle of the sprint:

- Change must be documented in the user story
- Impact on ETA must be explicitly stated in User story
- Task status must reflect the new scope
- Reopened Task with redefined *Original Estimate*

5.4 Reducing architect overload

Addresses RQ4: Chapter 4 revealed that architects are frequently used as **human coordination hub**. He is a main source of truth and helps to resolve tech implementation's small details' obstacles. This leads to constant clarification needs, overload, and less time for architectural work.

The proposed framework shifts from **people-based** to **system-based** coordination mechanisms:

- Tech task documentation answers most clarification questions
- Reduced ad-hoc communication, only in exceptional cases (stale tasks, blockers)
- Clear ownership and escalation paths

Under this framework the

- architect focuses on:
    - standards of development
    - cross-team tech design decisions
    - high-risk exceptions
- PMs rely on dashboards and alerts, not constant follow-ups
- Teams become self-sufficient in maintaining visibility

5.5 Enhancing visibility: what “good visibility” looks like

Supports RQ3 and RQ5: The last element of the framework outlines in practice what "good visibility" looks like. More meetings won't help with visibility; rather, accurate, timely, and practical information is what makes difference. The framework enables this through:

- **Azure DevOps dashboards** with
    - stale tasks (tasks without updates for N days)
    - blocker distribution (blocked tasks by role)
    - ETA value variance (planned vs actual)
- **Automation-driven alerts** with
    - highlighting exceptions
- **Consistent status semantics**
    - task’s status always shows real state of the work:
        - who is working on what
        - what is blocked and why
        - which tasks are at risk

- will likely finish within sprint

These framework help managers to spot early risks, prioritize interventions, and make wise decisions free from depending on unofficial channels of communication.

This chapter presents a cohesive Task Visibility and Predictability Framework that operates the insights from earlier chapters. By combining **process formalization**, **AI-assisted automation**, and **role clarity**, the framework addresses the root causes of unpredictability and visibility gaps identified in project. The next chapter focuses on how this framework can be realistically implemented and evaluated within a project.

CHAPTER 6 – IMPLEMENTATION PLAN

This chapter presents a structured implementation plan for the **Task visibility and predictability framework** proposed in Chapter 5. The purpose of plan is to translate qualitative findings and conceptual recommendations into a **practical, executable approach** that can be applied within the constraints of project.

6.1 Step-by-step implementation roadmap

The implementation is organized into five sequential phases, each addressing specific issues identified in Chapter 4 and aligned with the recommendations in Chapter 5.

Phase 1. Baseline alignment and process definition

The first phase emphasizes on building a common knowledge of the new process. The reason behind the framework is explained in a brief internal alignment session; it is linked to observed issues (stale tasks, ETA unpredictability, architect overload), and the fundamental ideas of structured updates are introduced.

During this phase, the following artifacts are defined:

- Standard task-status transition rules
- Minimum update requirements per status
- Agreed communication channel conventions (ADO vs Teams)
- Role-specific responsibilities

Phase 2. Process formalization in Azure DevOps

In this phase, the formal rules are embedded into Azure DevOps usage patterns. This includes:

- Clear definitions of when task status must be updated (event-based and time-based triggers)
- Mandatory comment for Dev testing evidence, blockers, investigations, and ETA changes
- Explicit use of “On Hold / Blocked” Tags to represent paused work

This phase directly addresses RQ2 by ensuring that task updates are no longer driven solely by personal habits.

Phase 3. Requirements and Definition of Done Standardization

Based on strong evidence from Chapter 4, unclear and changing requirements significantly reduce predictability. This phase introduces:

- templates of standardized acceptance criteria

- role-specific definition of done (DoD)
- change-handling rules for mid-sprint scope change

This step ensures that status updates and ETAs are grounded in a stable understanding of task completion.

Phase 4. AI-Assisted Automation Enablement

Automation mechanisms are introduced to reinforce – not replace – human discipline. These include:

- stale-task detection (e.g., no updates for N days)
- overdue ETA reminders
- blocker duration alerts

Notifications are delivered via Microsoft Teams using rule-based triggers. Importantly – the automation is configurable to avoid notification fatigue.

Phase 5. Monitoring, Feedback, and Adjustment

The final phase focuses on monitoring adoption and impact. The metrics (defined in Section 6.5) are reviewed regularly to identify friction points and adjust rules if needed. This phase will ensure continuous improvement.

6.2 Responsible of Roles (RACI matrix)

Clear responsibility allocation is critical to prevent reliance on informal coordination and reduce architect overload.

Activity	PM	Architect	Team Lead	Developer	QA	BSA
Define reporting rules	A	C	R	C	C	C
Maintain task updates	C	C	A	R	R	R
Monitor dashboards	R	R	C	I	I	I
Handle blockers escalation	A	R	C	R	R	C
Approve requirement changes	A	C	I	I	I	R
Automation configuration	R	C	C	I	I	I

Keys: R – Responsible (who does the work); A – Accountable (who owns the outcome); C – Consulted (who provides input); I – Informed (who is kept up to date)

Key principle:

The system – not individuals – becomes the primary coordination mechanism.

6.3 Timeline and deployment cadence

The framework is designed to be deployed within a 4–6 week window:

- Week 1: Alignment and process definition
- Week 2: ADO configuration and documentation updates
- Week 3: Requirements and DoD standardization
- Week 4: Automation activation (pilot mode)
- Weeks 5–6: Observation, tuning, and stabilization

This cadence minimizes delivery disruption while allowing behavioral adaptation**.**

6.4 Required tools

The implementation relies on tools:

- **Azure DevOps**
  - task tracking, comments, statuses, and evidence
  - task templates
  - dashboards and queries
- **Microsoft teams**
  - communication channel for reminders and alerts
  - bot-based nudges
- **Automation scripts / rules**
  - detection of stale tasks
  - time-based and event-based triggers

No machine-learning models or external analytics platforms are required.

6.5 KPIs, expected impact

The effectiveness of the framework is measured using **practical, management-relevant KPIs**, directly tied to the research questions.

Key Performance Indicators

- % of tasks updated within defined intervals
- Reduction in stale or long “In Progress” tasks
- Accuracy of ETAs (planned vs actual completion)

- Architect time spent on clarification
- Sprint predictability (User Stories / Tasks completed vs committed)

Expected Impact

- Enhanced predictability of task completion through constant updates
- Reduced architect participation in routine clarifications
- More openness and common knowledge among different roles.
- Less client-facing absentees from commitments

This chapter presented a realistic, orderly, and context-aware implementation plan for the suggested **Task visibility and predictability framework.** The framework directly solves the visibility and predictability issues found in IT projects by combining process formalization with AI-assisted automation, so remaining feasible within real-world constraints.

CHAPTER 7 – CONCLUSION

7.1 Summary of findings

This Capstone investigated the problem of **low task completion predictability and insufficient work visibility** in an outsourced IT project. Through qualitative interviews with delivery team – including Data Engineers, QA specialists, Business Analysts, and a Data Tech Lead – this study identified systemic weaknesses in how task progress is happened within Azure DevOps.

The findings demonstrate that the absence of **formalized task-update rules** leads to inconsistent reporting behaviors caused by individual habits, urgency, or external pressure rather than meaningful rules. As a result, task statuses and ETAs frequently lag behind reality, creating visibility gaps that undermine planning and decision-making. These gaps are further amplified by **frequent interruptions, context switching, and unplanned work**, which disconnect planned sprint commitments from actual execution.

The analysis also revealed a strong **overreliance on synchronous communication** – such as stand-ups, calls, and ad-hoc chats – to compensate for unreliable task data. While these practices temporarily restore situational awareness, they place a disproportionate coordination **burden on** project architect. Instead of focusing on architectural decision-making, he become central information hubs responsible for clarification and removing blockers across different channels.

Finally, the research findings indicate that participants shows **high readiness for improvement**. Most respondents expressed openness to **clearer process rules**, **standardized documentation expectations**, and lightweight automation – provided these mechanisms reduce manual overhead rather than increase bureaucracy. Collectively, the findings confirm that the problem is not a lack of effort or engagement, but a lack of **structured, enforceable, and supported task-visibility practices**.

7.2 Contributions to practice

This Capstone makes several practical contributions relevant to management of outsourced IT project.

First, it proposes a **Task Visibility and Predictability Framework** that integrates **process formalization** with **AI-assisted communication automation**. Unlike strictly procedural

approaches, the framework combines clear rules with automated reinforcement since it recognizes real-world constraints including interruptions, changing requirements, and limited attention.

Second, the study contributes a **role-clarified reporting model**, supported by a RACI matrix, that explicitly determining **who is responsible** for updating task status, **who is accountable** for data quality, and how **information** should **flow across roles**.

Third, the capstone demonstrates how **existing enterprise tools** – Azure DevOps and Microsoft Teams – can be configured to improve transparency without introducing custom software or complex analytics. By leveraging rule-based triggers, stale-task detection, and structured reminders can be enhance real-time visibility while minimizing additional process overhead.

Finally, the work provides **measurable operational indicators** – such as reduction in stale tasks, improved ETA accuracy, and decreased time spent in daily meetings – that managers and architect can use to evaluate the effectiveness of task-visibility improvements over time.

7.3 Limitations

This research has several limitations. First, it is based on a single outsourced IT project, which limits the generalizability of the findings to other projects within vendor. Second, the evaluation of the proposed framework is primarily qualitative; due to project constraints and NDA, large-scale quantitative was not conducted. Third, the research focuses only on the internal delivery team, excluding client-side perspectives that could further inform the impact of visibility gaps. Finally, the Capstone does not implement advanced machine-learning or predictive analytics, relying instead on rule-based automation and AI-assisted reminders in line with scope and practical constraints.

7.4 Future work

Several opportunities exist to extend this research in future studies.

One promising direction is the application of **machine learning and natural language processing (NLP)** techniques to historical task status updates data, comments, and teams communication logs.

Future research could also incorporate **client-side interviews and feedback**, enabling a more holistic assessment of how improved internal transparency influences client trust, satisfaction, and perceived delivery reliability.

Appendix A: Interview Questionnaire

Section A — Understanding Current Task-Update Practices

1. How do you currently update the status of your tasks in Azure DevOps?
2. What typically prompts you to update a task—your own decision, a reminder from the PM, or something else?
3. How often do you think tasks should be updated, and how often are they updated in practice?
4. What information do you normally include when updating a task (e.g., progress %, ETA, blockers)?
5. Do you feel the current update process is clear or ambiguous? Why?

Section B — Visibility of Work Progress

6. How clearly do you think the PM or architect can see your current workload and task status at any given time?
7. Are there moments when you believe your work is not visible to others? Can you provide examples?
8. How often do tasks remain "in progress" longer than expected without updated context?
9. What problems arise when task updates are delayed or incomplete?

Section C — ETA Accuracy and Task Completion Predictability

10. How do you estimate ETAs for your tasks? What factors make ETA accuracy difficult?
11. What typically causes tasks to take longer than initially estimated?
12. How comfortable do you feel providing ETA updates or revisions?
13. Do you believe ETA revisions are encouraged? Why or why not?

Section D — Blockers, Interruptions, and Communication Flow

14. How do you communicate blockers or issues that affect your task?
15. Do you feel blockers are escalated fast enough? Why or why not?
16. What communication channels do you use most often (Teams, comments, calls)?
17. Is the choice of communication channel clear, or does it depend on individual preference?

Section E — Role Clarity and Coordination Overload

18. Whom do you approach when you need clarification on a task?
19. How often does the architect intervene in your work or task clarifications?
20. Do you feel the architect or PM is overloaded with unnecessary communication?
21. What types of questions or issues should be handled through standardized updates instead of direct messages?

Section F — Perception of Project Management and Sprint Execution

22. From your perspective, how predictable is task completion in the current sprint process?
23. What makes sprint planning or mid-sprint adjustments challenging?
24. Do you feel the team has visibility into who is working on what, and how much progress has been made?

Section G — Pain Points and Current Inefficiencies

25. What frustrates you the most about the current task-tracking and communication process?
26. Can you describe a recent incident where lack of visibility caused delays or rework?
27. What are the most common reasons that task updates are missing or incomplete?

Section H — Readiness for Formalization and Structured Processes

28. Do you believe formalizing the reporting process would help or hinder your work? Why?
29. What specific rules or routines would make your task updates easier and more consistent?
30. What concerns do you have about introducing mandatory update protocols?

Section I — Attitudes Toward Automation (AI-Assisted Communication)

31. Would automated reminders help you keep tasks updated on time? Why or why not?
32. Would you find value in a Teams bot prompting for status updates or ETAs?
33. Which automated notifications would be most helpful (e.g., stale tasks, blockers, overdue ETAs, review reminders)?

Appendix B: Raw Data

Q1 After a task is assigned, I validate its priority and transition it to "In Dev." Upon completing the implementation, I move it to "Dev Done." I then perform development testing, verify the functionality in both the Dev and QA environments, and finally transition the task to "Ready for QA."

Q2 task progress and its criticality”

Q3 Tasks are updated according to the progress made; the number of updates can be unlimited.

Q4 All task-related information must be attached, including requests, current data outputs, questions, and any blockers.

Q5 I think the process is much clearer than it was, for example, a year ago.

Q6 At the moment, the two stand-ups give a fairly accurate picture of what each developer is working on.

Q7 No such examples exist.

Q8 It all depends on the context. For example, there is a bug on a specific item — the root cause is identified, but during the investigation, other inconsistencies may surface that can affect other items and results. As a result, the fix may be delayed due to calculations or dependencies on other items, and a single bug can end up including 2–3 fixes.

Q9 Each case needs to be considered separately. In practice, when Dev Testing is performed, anomalies are discovered and an investigation begins. Depending on errors in the BE, DE, or FE, the situation is completely different each time.

Q10 I wanted to understand the complete business context — the rationale behind each coefficient, its calculation, and its purpose, so that it can be clearly explained. Afterward, the task is broken down into actionable steps and approved.

Q11 There are many factors involved: task switching, changing conditions, and a larger workload than initially expected.

Q12 Considering that demos of completed work happen quite frequently, even a small delay of 3–4 hours can create tension, so it’s necessary to communicate delays in advance.

Q13 Revisions of ETA are inevitable, because obtaining an accurate ETA requires either doing simple tasks or tasks without any investigate

Q14 I report who is blocking me, and if the fix will take a significant amount of time, I raise the issue during the stand-up.

Q15 There are no issues with this; the response is very fast.

Q16 I first attempt live communication via chat or calls. If the issue is not resolved promptly, I document it by leaving comments.

Q17 I use any available communication channel.

Q18 To all individuals who have knowledge to assist with this.

Q19 For user stories, quite often; for bugs, less frequently.

Q20 No, the more communication with developers, the better.

Q21 It all depends on the criticality. If there is time, I leave comments and wait for responses; if not, I write in the chat.

Q22 If the task is large, the likelihood is low, since we are developing a unique product and it is impossible to foresee all aspects.

Q23 Unforeseen and unplanned tasks or changes during the process.

Q24 I'm always curious about who is working on what; I usually check pull requests.

Q25 There's nothing particularly special; it's much better now than it was a year ago.

Q26 Limited time for adequate testing and verification.

Q27 Delivery deadlines

Q28 If formalization takes development time, then yes, it may interfere.

Q29 I don't have any specific ideas.

Q30 No concerns.

Q31 They won't be unnecessary.

Q32 Let's try this approach.

Q33 Reminders are acceptable as long as their quantity is limited.

References

1. Alyahya, S. (2013). Agile software development in distributed environments: A systematic review and empirical investigation (Doctoral dissertation, Cardiff University).
2. Baiyere, A., Salmela, H., & Tapanainen, T. (2021). Digital transformation and the new logics of business process management. arXiv. https://arxiv.org/abs/2106.06154
3. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). Manifesto for Agile Software Development. https://agilemanifesto.org/
4. Creswell, J. W., & Poth, C. N. (2018). Qualitative inquiry and research design: Choosing among five approaches (4th ed.). SAGE Publications.
5. Gartner. (2023). Improving engineering productivity through workflow automation. Gartner Research.
6. Hoda, R., Noble, J., & Marshall, S. (2013). Self-organizing roles on agile software development teams. IEEE Transactions on Software Engineering, 39(3), 422–444. https://doi.org/10.1109/TSE.2012.30
7. Kerzner, H. (2022). Project management: A systems approach to planning, scheduling, and controlling (13th ed.). Wiley.
8. KindOPSTAR. (2023). QualiGPT: AI-assisted qualitative analysis tool [GitHub repository]. https://github.com/KindOPSTAR/QualiGPT
9. Microsoft. (2024). Azure DevOps documentation. https://learn.microsoft.com/en-us/azure/devops/
10. Microsoft. (2024). Azure DevOps Analytics OData reference. https://learn.microsoft.com/en-us/azure/devops/report/odata/
11. Microsoft. (2024). Microsoft Teams platform documentation. https://learn.microsoft.com/en-us/microsoftteams/platform/
12. Open coding in grounded theory (+ example). (n.d.). Shribe. https://shribe.eu/open-coding/
13. Open coding in qualitative research. (2024). ResearchGate. https://www.researchgate.net/publication/387687208_Open_Coding_In_Qualitative_Research

14. PMI. (2021). A guide to the project management body of knowledge (PMBOK® Guide) (7th ed.). Project Management Institute.
15. Redefining qualitative analysis in the AI era: Utilizing ChatGPT for efficient thematic analysis. (2023). arXiv. https://arxiv.org/abs/2309.10771
16. Schwaber, K., & Sutherland, J. (2020). The Scrum Guide. https://scrumguides.org/
17. Sedlmair, M., Meyer, M., & Munzner, T. (2013). Information visualization for agile software development. IEEE Computer Graphics and Applications, 33(1), 26–35.
18. Sharp, H., Robinson, H., & Woodman, M. (2000). Software engineering: Community and culture. IEEE Software, 17(1), 40–47. https://doi.org/10.1109/52.819971
19. Strauss, A., & Corbin, J. (1998). Basics of qualitative research: Techniques and procedures for developing grounded theory (2nd ed.). SAGE Publications.
20. Tenhunen, M.-L., Cantemir, D., & Christian University. (2023). Enhancing decision-making with artificial intelligence in project management. Dimitrie Cantemir Christian University.
21. Too, E. G., & Weaver, P. (2014). The management of project management: A conceptual framework for project governance. International Journal of Project Management, 32(8), 1382–1394. https://doi.org/10.1016/j.ijproman.2013.07.006
22. Zhang, Y., Mockus, A., Keivanloo, I., & Bird, C. (2022). Towards reliable work item estimation: An empirical study. IEEE Transactions on Software Engineering, 48(8), 2751–2766. https://doi.org/10.1109/TSE.2021.3057829