Reducing Uncertainty in Large-Scale Agile Development Projects

Vincent I. Prado Claudio
Master in Engineering Management
Dr. Héctor J. Cruzado
Graduate School
Polytechnic University of Puerto Rico

Abstract *— Managing uncertainty in geographically dispersed Agile software development projects remains a challenge. This paper explores methods implemented by a large-scale defense contractor team using Scrum methodologies. The project aimed to improve predictability and reduce effort estimation errors by focusing on user story writing and sprint planning practices. A key finding was the importance of establishing clear definitions for "ready" and "done" user stories, incorporating a modified INVEST method, and utilizing story writing templates. To address uncertainty during sprint planning, scales were devised to measure technical uncertainty and Subject Matter Expert (SME) availability. A decision matrix was created based on these scales to guide story inclusion within a sprint. The implemented guidelines have the potential to enhance communication, reduce rework, and improve overall project predictability for geographically dispersed Agile teams.*

Key Terms *— Definition of "Done", Definition of "Ready", INVEST method, Scrum.*

INTRODUCTION

This paper addresses managing uncertainty in software development for a geographically dispersed defense contractor team. The team, composed of software and system engineers working remotely across Puerto Rico and the US, utilizes the Scrum agile development methodology. Scrum emphasizes adaptability through iterative development cycles known as sprints. This project aimed to improve the team's Scrum practices to better manage these uncertainties. It focused on two key areas: user story writing and sprint planning.

Despite widespread adoption, Agile methodologies present practical challenges. A significant hurdle arises from the lack of comprehensive upfront requirements compared to Waterfall methods. This ambiguity can lead to underestimating task complexity and, as experienced by the project team, force the withdrawal of a month's worth of work. To mitigate these issues, this project aimed to decrease development process uncertainty, with the goal of improving task completion time estimates and facilitating the creation of more predictable sprint cycles.

BACKGROUND

The growing complexity of software projects, coupled with a surge in demand for new applications, has forced companies to rethink their development methodologies. Traditional approaches often struggle to adapt to these challenges. In response, many software teams are turning to agile development approaches, which offer greater flexibility in handling requirement changes and prioritize meeting customer needs while achieving rapid time-to-market [1]. While agile is often associated with smaller projects, its iterative development cycles prove valuable in managing uncertainty, a common issue that can stem from various factors beyond technical experience or poorly defined requirements [2]. Software complexity, stringent quality demands, and tight schedules can all contribute to project uncertainty [3].

Factors that can significantly impact project uncertainty, include market uncertainty (e.g., evolving customer needs, competitor actions), technical uncertainty (e.g., feasibility of new technologies, integration challenges), project duration (e.g., tight deadlines), and external dependencies (reliance on other teams or external resources). The ability to categorize uncertainty allows project managers to select appropriate

development approaches [4]. Agile methodologies are particularly well-suited for projects with high uncertainty due to their iterative nature and adaptability. Agile's focus on continuous feedback and incremental delivery enables teams to adjust to changing requirements and unforeseen challenges, ultimately reducing the negative impact of uncertainty on project outcomes.

Two crucial areas that significantly influence the management of uncertainty in software development projects are knowledge sharing and requirement engineering. Requirement engineering is a systematic approach to defining, documenting, and verifying stakeholder needs, ensures a clear understanding of expectations and translates them into actionable requirements [5]. Effective knowledge sharing within the development team strengthens this process. For instance, communication barriers in large teams can hinder requirement engineering efforts [6], particularly with the challenges of remote work environments [7]. Requirement uncertainty often arises at the project's outset due to incomplete or unclear information from clients, management, and developers [8]. While Agile methodologies aim to reduce the impact of change and uncertainty [1], inadequate requirement analysis and poor knowledge sharing can still lead to project difficulties.

Agile methodologies offer several practical strategies for managing uncertainty. Furthermore, adjusting iteration length can be beneficial. Shorter iterations are typically recommended when there's high uncertainty, as they enable faster feedback loops [9]. This allows teams to address requirement ambiguities and correct course quickly, thereby reducing uncertainty. Conversely, longer iterations can be suitable for well-defined projects with minimal knowledge gaps. Additionally, fostering knowledge sharing within the team is crucial. Team members can leverage various methods like presentations, code reviews, or pair programming to share expertise in specific areas [7]. This collaborative learning approach can significantly reduce knowledge gaps, a major source of uncertainty. Finally, tackling the most difficult parts of a project first can also be advantageous [2]. By prioritizing tasks with high learning value, team members gain essential knowledge early on, enabling them to create well-defined requirements and navigate unforeseen challenges with greater confidence.

Effective user story writing hinges on establishing clear definitions for "ready" and "done" [10][11]. The "ready" criteria determine when a story is prepared for development. This might include approval from the product owner and team consensus on the requirements. Conversely, the "done" criteria specify when a story is complete. Furthermore, to enhance user stories, the INVEST method is highly recommended [12][13]. INVEST serves as an acronym for a set of guiding principles: Independent, Negotiable, Valuable, Estimable, Small, and Testable. Independent stories can be tackled without waiting for others to finish. Negotiable stories offer flexibility for exploring different implementation approaches. Valuable stories demonstrably benefit the product, customer, and business. Estimable stories allow the team to gauge the time and effort required. Small stories can be finished within a reasonable timeframe. Finally, Testable stories outline clear success criteria for the team to verify completion.

It should be noted that there has not been much work done on the use of Agile for large scale projects [1]. By investigating successful strategies for managing uncertainty in similar contexts, guidelines were formulated to best adapt agile methodologies for the specific needs of the defense company's large-scale software development project.

METHODOLOGY

The initial phase of this methodology involved data collection and analysis from user stories. The objective was to identify recurring patterns that contribute to ambiguities within the stories. The analysis focused on key factors including

acceptance criteria, technical complexity, estimated completion time, and the author's area of expertise.

Different factors were found to contribute towards uncertainty; however, the main factor contributing to uncertainty was the inconsistent user story format. While some stories shared a basic structure, the team lacked a standardized format for user story writing. Over time, this resulted in increased deviation in story composition, leading to the omission of crucial details necessary for comprehensive understanding. For instance, some bugs fix stories solely described the issue but neglected to outline the expected behavior after the fix.

The second phase involved a literature review focused on knowledge sharing, requirement uncertainty, and Scrum-based user story writing. This review aimed to glean insights that would later inform the development of uncertainty-reduction guidelines. Interestingly, the literature revealed a dearth of information on quantifying uncertainty, with some exceptions. Likewise, there was a lack of guidance on user story writing within the context of large-scale projects. Despite these limitations, the available resources proved sufficient to generate new ideas and propel the project forward.

The methodology culminated in the formulation of conclusions based on the findings from the initial two steps. Subsequently, techniques and guidelines were established to mitigate uncertainty within the team's development process.

RESULTS

Analysis of numerous user stories and insights gleaned from the literature review revealed the team's lack of standardized definitions for "ready" and "done" within the Scrum framework. To address this gap, two distinct user story formats were developed, tailored to either feature implementation or bug fixes. Table 1 outlines the mandatory sections for feature-oriented stories, while Table 2 details the required sections for bug reports. Both tables utilize a three-column structure: the first column specifies the section name, the second column describes the expected content, and the third column provides guiding questions for the author. These questions ensure the story fulfills its intended purpose. A "ready" story, according to the established definition, must have answers to all the listed questions within the corresponding sections.

Table 1
Point Sizes and Type Styles

Section Name	*Section Description*	*Guiding Question*
Description	This part will explain what the problem is. It does not have to contain the expected behavior.	What is the problem?
Expected Changes	Explain what the expected result is. This section should be part of the “definition of done”.	What should happen to confirm the issue was fixed?
Steps to Reproduce	List steps to reproduce the issue. Make sure to include the dependencies.	What are the steps that should be followed to reproduce the issue?

Table 2
Story Sections for Implementation of New Features

Section Name	*Section Description*	*Guiding Question*
Description	Explain why the story is needed.	What benefit will it provide for the end-user?
Expected Changes	Explain what the expected result should be.	What new capabilities do these changes add to the system? What will the user be able to do that was not able to do in previous versions of the system?
Documentation	Add links to any documentation and/or design that was created for this new feature or any related story that was previously worked.	Has any previous work been done for this feature? Were designs and/or documentation created for this feature?

Implementing story writing templates promotes consistency and streamlines the process for product owners. Templates facilitate story review and question formulation prior to sprint planning. Additionally, they aid in identifying inconsistencies when evaluating stories against the "ready" criteria. A story missing essential elements from its corresponding template fails to meet the "ready" definition and should not be considered for development until complete.

In the context of this large-scale project, a specific definition of "ready" for user stories was established. This definition comprised two key criteria: comprehensive content and INVEST compliance (with adaptation). All stories must incorporate every element outlined in the designated story templates. While the INVEST principles are generally recommended, the "independent" and "negotiable" aspects were deemed less crucial due to the project's scale. The complexity necessitated dividing extensive features into smaller, dependent stories. In some instances, this approach even proved advantageous for the team.

The team's definition of "done" for user stories encompasses two primary aspects. First, all modifications outlined in the "expected changes" section of the story must be implemented. This ensures the core functionality of the story is complete. Second, the definition encompasses any additional steps required to finalize the story, even if not directly related to the addressed feature or issue. In this team's workflow, these supplementary steps encompass system verification and code or design reviews.

Another challenge faced by the defense contractor team involved determining an appropriate threshold for high-uncertainty stories within a sprint. To address this concern, two measurement scales were devised: one to quantify story uncertainty shown in Table 3 and another to gauge Subject Matter Expert (SME) availability shown in Table 4. Given the project's scale, direct client interaction is limited for the team. To bridge potential domain knowledge gaps, Subject Matter Experts (SMEs) within the team play a crucial role. Their availability becomes particularly critical during sprints containing high-uncertainty stories.

Table 3
Scale for Scoring Technical Uncertainty

Attribute	*Uncertainty Score*
Enhancement	1
Know how to build	2
Not sure how to build but have previous domain experience	3
Not sure how to build and no domain experience	4
Research is needed (New technology and/or no design)	5

Table 4
Scale for Scoring SME Availability

Attribute	*Uncertainty Score*
Enhancement	1
Know how to build	2
Not sure how to build but have previous domain experience	3
Not sure how to build and no domain experience	4
Research is needed (New technology and/or no design)	5

To address uncertainty and SME availability during sprint planning, a decision matrix shown in Figure 1 was created. This matrix utilizes the established scales for technical uncertainty in Table 3 and SME availability in Table 4 to guide story inclusion within a sprint. Prior to sprint commencement, SME availability is determined. Subsequently, each story's technical uncertainty is measured to locate its corresponding position on the matrix. The matrix offers five potential actions for each story:

- **Research Story:** Create a preliminary research story to be completed before tackling the current story.
- **Weekly Meetings:** Schedule dedicated weekly meetings to address uncertainties.
- **Story Splitting:** Divide the story into smaller, more manageable units.
- **Removal:** Remove the story from the current sprint plan.

- **Standard Inclusion:** Include the story in the sprint as planned

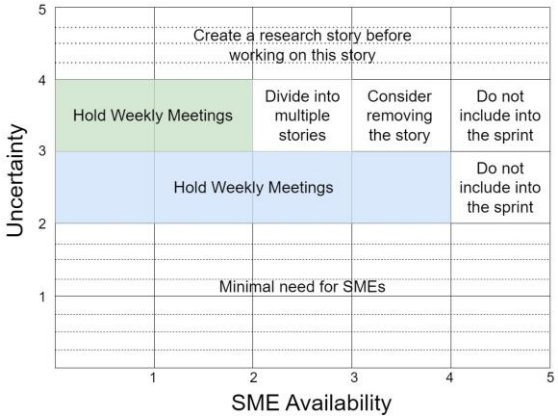

Figure 1
SME availability vs Technical Uncertainty Matrix

CONCLUSION

This work presented guidelines developed to address uncertainty in the software development process of a geographically dispersed defense contractor team utilizing Scrum methodologies. The project focused on user story writing and sprint planning practices to enhance predictability and reduce estimation errors. Key findings included the importance of establishing clear definitions for "ready" and "done" user stories, incorporating the INVEST method with adaptations for large-scale projects, and utilizing story writing templates to ensure consistency. Scales to measure technical uncertainty and Subject Matter Expert (SME) availability were created to inform sprint planning decisions. The implementation of these guidelines has the potential to improve communication, reduce rework, and enhance overall project predictability within the defense contractor team.

REFERENCES

[1] Petersen, K, et al., "A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. "*Journal of systems and software* 82.9, 2009, pp.1479-1490.

[2] Racheva, Z, et al., "Do we know enough about requirements prioritization in agile projects: insights from a case study." *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 2010, pp. 147-156

[3] Levy, M, et al., "Knowledge management in practice: The case of agile software development." *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering. IEEE*, 2009, pp. 60-65

[4] Little, Tu, "Context-adaptive agility: managing complexity and uncertainty.", *IEEE software*, 22.3, 2005 pp. 28-35.

[5] Jin, Zhi. Environment modeling-based requirements engineering for software intensive systems. Morgan Kaufmann, 2017.

[6] Rasheed, A, et al. “Requirement engineering challenges in agile software development." *Mathematical Problems in Engineering* 2021, 2021, pp. 1-18.

[7] Dorairaj, S, et al., "Knowledge management in distributed agile software development.", *2012 Agile Conference*, IEEE, 2012, pp. 64-73

[8] Haleem, M, et al, "Tackling requirements uncertainty in software projects: a cognitive approach." *International Journal of Cognitive Computing in Engineering* 2, 2021, pp. 180-190.

[9] Highsmith, J. “Adaptive software development: a collaborative approach to managing complex systems”, Addison-Wesley, 2013, pp.95

[10] Atlassian, *Definition of Ready (DoR) Explained & Key Components*, Available: https://www.atlassian.com/agile/project management/definition-of-ready

[11] M. Rehkopf, *User stories with examples and a template*, Atlassian, Available: https://www.atlassian.com/agile/project-management/user-stories

[12] U.S. General Services Administration, *Writing effective user stories*, Available, https://tech.gsa.gov/guides/effective_user_stories/

[13] K. Drozd, *How to use Confluence and Jira Software for agile sprint planning & refinement*, Atlassian, Available: https://www.atlassian.com/agile/tutorials/jira-confluence-sprint-refinement