A Comparison of Requirements Management in Agile vs. Predictive Projects

Dallas Rosson
Naval Undersea Warfare Center
Division Keyport
Keyport, WA
Industrial and Systems Engineering
University of Washington
Seattle, WA
dallas.j.rosson.civ@us.navy.mil

Daniel R. Herber
Systems Engineering
Colorado State University
Fort Collins, CO
daniel.herber@colostate.edu

Alexandra Trani
Naval Undersea Warfare Center
Division Keyport
Keyport, WA

Thomas H. Bradley
Systems Engineering
Colorado State University
Fort Collins, CO
thomas.bradley@colostate.edu

Abstract. Agile and *Predictive*-style approaches each have their own pros and cons, managing project requirements in different, often conflicting ways. Applying these styles independently to critical software systems can result in adverse effects. Agile-style requirements management, typically documented in Connextra-style User Stories, offers customers and stakeholders the flexibility to adapt requirements as projects evolve. The highly detailed and structured *Predictive*-style requirements management, often documented in INCOSE style "shall" statements, provides clear visibility into system functions and supports communication with developers. Understanding the benefits and limitations of each style equips developers with an expanded toolset, increased perspective, and the ability to apply each style more effectively to projects. By thoughtfully combining the strengths of both requirements management styles, a more comprehensive requirements baseline can be established, minimizing risks for both customers and programs.

Keywords. Agile, Software Systems Engineering, Scrum, Software Requirements

Introduction

There are significant differences between *Agile* and *Predictive* styles of project management. Projects managed with iterative and flexible *Agile* methods focus on customer needs and desires without a high burden of documentation and planning. Because flexibility is prioritized, *Agile* projects can quickly pivot in response to changing customer needs. In general, *Agile* practitioners rely on "stories" in lieu of requirements to communicate customer needs to development teams.

In contrast, projects managed with *Predictive* methods (e.g., the Waterfall methodology) rely on extensive upfront planning to capture customer needs, goals, and requirements in documentation. Documentation is created with the objective of clear communication and effective capture of project scope. The Department of Defense (DoD), along with many other groups and companies in the software engineering industry, are increasingly adopting *Agile* project methodologies. Yet, as this article demonstrates, there are cases where traditional, *Predictive* project management remains ideal. This article will illustrate the benefits and drawbacks of each requirements management framework for DoD software development projects by comparing requirements management, traceability transformation, and goals to requirements refinement.

This paper is organized in seven sections. Section 2 and 3 begin with an introductory overview of the *Agile* and *Predictive* development methodologies. Section 4 presents a comparison of requirements management in *Predictive* versus *Agile* projects, discussing the benefits and drawbacks of each. A comparison of requirements traceability is explored in Section 5. In Section 6, we argue that *Agile* requirements, or User Stories, are actually goals and do not meet the definition of traditional requirements. Finally, we conclude in Section 7 by presenting the need for a system capable of combining the strengths of both the *Agile* and *Predictive* methods.

Overview of *Agile* Software Development

Agile project management is preferred in software development projects for its flexibility (Fowler and Highsmith, 2001; Grenning, 2017). Using these principles, multiple ways of executing *Agile* projects have been developed including Crystal, eXtreme Programming, and Scrum (Ashmore and Runyan, 2014). *Agile* practices have further evolved to program and business management through the utilization of scaling *Agile* with methods such as Scrum of Scrums (SoS) or the *Scaled Agile Framework* (SAFe) (Hohl et al., 2018).

Agile project management is most useful for high-risk, low-certainty projects in which the requirements are only partially known (Project Management Institute, 2017; Wysocki, 2012). *Agile* reduces project risk through incremental delivery of developed capabilities in conjunction with frequent customer interactions and demonstrations (Project Management Institute, 2017; Schwaber and Sutherland, 2020). As shown in Figure 1, software development teams deliver minimum viable products to customers in small, iterative increments, re-prioritizing work backlogs at each iteration, and utilizing a customer feedback loop to integrate changes driven by learning generated from the delivery of product increments. This iterative and incremental approach helps to lower risk and optimize predictability through customer and subject matter expert engagement with the development team (Schwaber and Sutherland, 2020). By focusing on the most important need of a

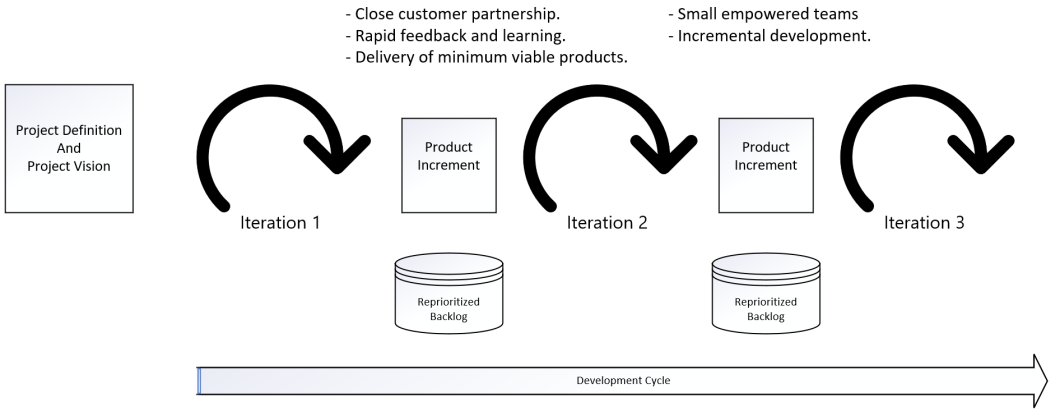

Figure 1. Adaptive Development Approach

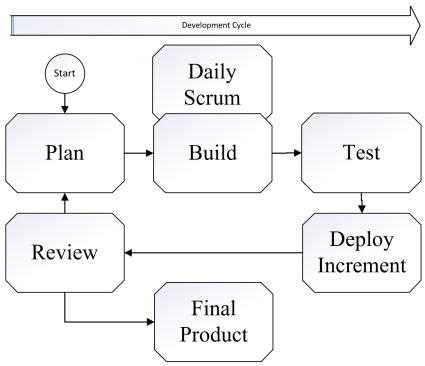

Figure 2. Example *Agile* Scrum Development Lifecycle

customer and then iteratively revisiting the project requirements, *Agile* methods seek to make use of Lean thinking, reducing waste and focusing on what the customer needs, lowering overhead burdens (Hines et al., 2004).

Knowledge work (e.g., planning, scheduling, resource allocation) is difficult for humans to perform within complex systems (Davenport, 2005). Scrum, one of the most widely used forms of *Agile* program management (Paulk, 2013; Rauf and Al Ghafees, 2015), assists with this effort by optimizing heuristics, providing continuous feedback, and creating external, cognitive artifacts like sticky notes, models, and the Scrum Board (Verwijs, 2020; Verwijs and Russo, 2023). Figure 2 shows an example of the Scrum product lifecycle. This lifecycle is a constant loop, progressing through the closed loop stages of Plan, Build, Test, Deploy Increment, and Review. During the Build Stage, a Daily Scrum activity occurs. After the Review stage, a Final Product might be produced if the Product Owner deems the product ready for production. This execution loop occurs each Sprint and continues until project completion.

Through this iterative process, *Agile* software development focuses on lightweight, flexible, and adaptable project execution. *Agile* Scrum further codifies the broader *Agile* methodology by implementing a structured framework that includes an iterative development cycle called a Sprint. *Agile* Scrum practitioners plan projects in Sprint increments, modifying execution plans as customer needs and requirements change. Through the utilization of *Agile* project management tools, *Agile* practitioners can adapt to project with minor impact on cost, schedule, and performance.

Overview of the *Predictive* Development Methodology

Traditionally, project management is predicated on a *Predictive* lifecycle (Wysocki, 2012).*Predictive* life-cycles are expected to have little flexibility or change over the course of execution, have readily available and stable team members, and accept negligible risk (Project Management Institute, 2017). These projects are most successful when "project and product requirements can be defined, collected and analyzed at the start of a project" (Project Management Institute, 2021). This type of project management is well-suited for construction, with a repeatable and sequenced progression of work (Project Management Institute, 2017; Project Management Institute, 2021). An example *Predictive* lifecycle flow is shown in Figure 3 where project phases are executed in a serial manner, starting with the Plan phase and working through all phases until the Deploy phase is completed. Many modern software projects have evolving project and product requirements, are high risk, and have significant uncertainty, making it hard to apply the *Predictive* development approach to these types of projects successfully (Project Management Institute, 2017; Project Management Institute, 2021; Wysocki, 2012).

Figure 3. Example *Predictive* Lifecycle

The *Predictive* methodology of project management is a linear and serial effort. The previous phase must be completed prior to moving to the next phase (Project Management Institute, 2021). Phases are not repeated, and work products should not be touched upon again (Wysocki, 2012). The only feedback loop that is present in this type of model normally occurs during the Test portion of the lifecycle. Defect fixes and changes can be approved and performed during this period. One of the major issues with this linear model is that when changes are requested or defects occur, there is a resultant extension of the project completion date, which creates Earned Value compromising problems (Wysocki, 2012).

When applied to software projects, *Predictive* development has many documented issues. Due to the strict adherence in an attempt to "provide straightforward, systematic, and organized processes" through detailed planning and documentation, it commonly fails to adapt to the "rapidly changing business requirements", a focus on project metrics, such as Earned Value, and less of a focus on flexibility in execution for process improvement (Flora and Chande, 2014). For the *Predictive* approach to work, a large emphasis on apriori planning, measurement, and control is put in place. If very little is expected to change throughout a project or when a product is difficult to change after development execution begins, this approach is a very good use of time resources (Project Management Institute, 2021). Unfortunately, modern software development projects frequently change and are relatively inexpensive to alter after development. Thus, when compared to *Agile* methods, *Predictive* methods lack needed flexibility in project execution and prove more costly due to larger overhead and planning burdens. Metrics of project performance for *Predictive* projects are often not a reflection of reality, with the most common being Earned Value, lacking the ability to track software execution and health (Project Management Institute, 2017). As a result, most software development projects have moved away from the traditional, *Predictive* project management methodology.

Requirements management is different across *Predictive* and *Agile* projects. *Predictive* projects generate detailed requirements documents that are baselined prior to the start of development efforts (Project Management Institute, 2021). *Agile* projects use more flexible requirements, generally gathered at a very high level, to form an idea of what a project goal should be prior to the start of development (Project Management Institute, 2017). *Predictive* focuses on having a solid plan and sticking to it, whereas *Agile* focuses on flexibility in execution and welcoming changes (Fowler and Highsmith, 2001; Project Management Institute, 2017; Project Management Institute, 2021).

Predictive and *Agile* methods are dramatically different and are best applied in different scenarios. *Predictive* methods are best suited to projects with minimal risk of change and uncertainty for which a detailed requirements document can be generated. In these cases, detailed planning can lead to cost savings through the avoidance of unnecessary schedule slippage and cost planning. *Predictive* approaches are most suited to applications such as manufacturing and construction, where the final product is known, tolerances are documented, and the customer has a low likelihood of presenting changes to requirements (Project Management Institute, 2021; Wysocki, 2012). *Agile* methods are suited for medium to high risk projects with a high degree of uncertainty where it is easy to pivot and course correct in the middle of project execution. In these types of projects, requirements do not need to be fully specified prior to the start of development. Instead, requirements can be elicited and documented at a higher level where customer needs and goals are noted but not necessarily down to the atomic level (Fowler and Highsmith, 2001; Project Management Institute, 2017; Wysocki, 2012). This is especially applicable in the software development industry, as producing code has

little overhead cost outside of labor. Preplanning is not required for these projects nor are there concerns of retooling, logistics and supply issues.

A Comparison of Requirements Management in *Predictive* versus *Agile* Projects

Predictive-style requirements are described in IEEE and INCOSE documentation. IEEE stipulates that requirements statements "translate or express a need and its associated constraints and conditions" and are "written in a language which can take the form of a natural language" (IEEE, 2011). INCOSE further defines requirements utilizing specified language, such as a "shall" statement (INCOSE Requirements Working Group, 2023). These requirements are very detailed and must meet the following characteristics: necessary, appropriate, unambiguous, complete, singular, feasible, verifiable, validatable, correct, and conforming (INCOSE Requirements Working Group, 2023). Furthermore, the set of requirements must adhere to the following characteristics: complete, consistent, feasible, comprehensible, validatable, and correct (INCOSE Requirements Working Group, 2023). Requirements management in *Predictive* projects is, therefore, a detailed and large undertaking. The goal of an initial baseline requirements set is to have a fully formed vision and understanding of the proposed system.

In contrast, *Agile* DoD projects usually generate requirements in the form of Connextra User Stories. The Connextra form of User Story is written as the following: "As a role, I want goal, [so that][benefit]" (Fowler and Highsmith, 2001). These User Stories became popular through the ideas of eXtreme Programming, where development teams use User Stories to "understand the flow of requirements from initial ideas to final product" (Hines et al., 2004). These User Stories are stored by the developers in a backlog where they are prioritized by a Product Owner who sets priority according to their understanding of customer needs and desires (Das et al., 2021).

User Stories are tracked via a system called "Story Points". Story Points "rate the relative work, risk, and complexity of a requirement or story" (Project Management Institute, 2017). As it is difficult to know exactly how much work, risk, or complexity a work item will have without breaking the item down into an atomic, easily testable, and understandable state, this can be a difficult task to accomplish. Many teams use a form of Story Pointing called Planning Poker to form a level set baseline for Story Point values (Das et al., 2021; Mountain Goat Software, 2024). Each team has a different method to set point equivalency, and it can take time for the point values to stabilize with a new team. Experienced developers are of great assistance in this aspect, and it is recommended that teams be made up of junior and senior developers, as a study performed by Mahnic et al. found that "optimism bias caused by group discussion diminishes or disappears by increasing the expertise of people involved in the group estimation process" (Mahnic and Hovelja, 2012). Furthermore, the complexity of estimation increases as groups can become over-confident in the accuracy of their estimates when given tasks more difficult than are usually presented. Software developers often have problems predicting their own capability of work, and overall experience can be very "narrow" (Jorgensen, 2004). Therefore, while Story Pointing efforts can be effective in an experienced and fully realized team with mixed experience levels, there are valid concerns regarding the validity of *storming to norming* teams and their ability to estimate software project workloads adequately.

So, although both *Agile* and *Predictive* methods use these techniques to develop and communicate customer intent, in practice, they are not equivalently effective in various project contexts. For example, 46% of customers report *Agile* projects as "unsuccessful" within the boundaries of client benefits, cost control, and time control (Azanha et al., 2017). Furthermore, for *Agile* projects to be successful, customers must actively participate in the process and are expected to be "Collaborative, Representative, Authorized, Committed, and Knowledgeable" (Atlassian, 2023). Ineffective communication and poor requirements development are driving factors for failure in *Agile* projects (Bijan et al., 2013).

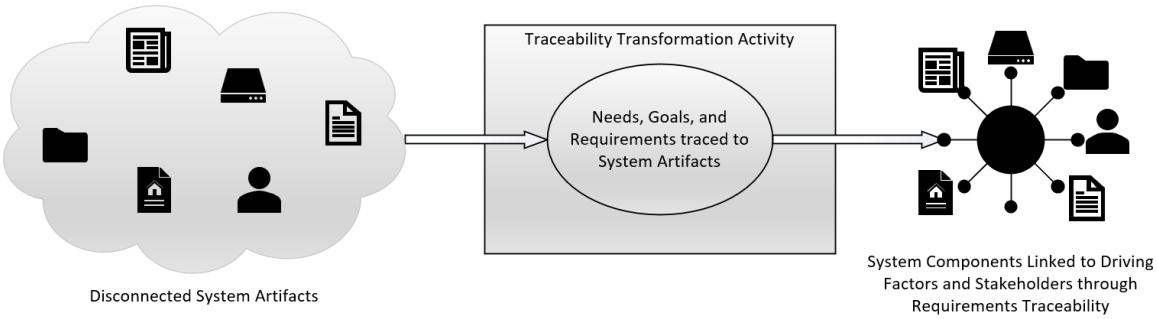

Figure 4. The Transformation of Disconnected System Artifacts to a Requirements Traced System

The use of thorough, standardized, and atomic requirements in the form of *Predictive*-style ”shall” statements increases the ability of developers and customers to communicate with one another, directly leading to improved system and problem domain understanding (Rosson, 2024). Especially in the instantiation phases of projects where Needs, Goals, and Requirements are generated – even if they are solely high-level and abstract. Because of the limited and abstract nature of User Stories, their use can lead to vague requirements as well as confusion within *Agile* project development teams. Problems in cost, schedule, and performance are even more likely to arise as availability of stakeholders is reduced.

A Comparison of Requirements Traceability

Introduction to Requirements Traceability

Requirement owners must trace requirements to needs for project success (Dove et al., 2023). Requirements traceability is defined as “the ability to describe and follow the life of a requirement in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)” (Chief Information Officer, 2010). Mature projects include traceability not only during one point in time, such as between customer requirements and system requirements, but also throughout time as requirements change and evolve (Boehm, Lane, et al., 2007). Through the application of traceability, the disparate artifacts that encompass a system are connected through linkages to other system components. Figure 4 shows how individual and disconnected system artifacts flow through a transformation activity where needs, goals, and requirements are traced directly to the artifacts, resulting in a network of system components. Maintaining traceability of system artifacts to individual requirements creates greater visibility and understanding of the system.

Agile Requirements Traceability

Agile practitioners recognize the importance of requirements traceability as a key enabler in maintaining customer focus during the software development process (Lee et al., 2003). Yet, requirements traceability is not inherent in *Agile* Scrum and is not mentioned in the Scrum Guide, *Agile* Software Development, or the *Agile* Practice Guide (Dingsoyr et al., 2010; Hines et al., 2004; Project Management Institute, 2017). *Agile* projects generally track requirements at four levels: Epics, Features, User Stories, and Tasks. Traceability utilization beyond the hierarchy prescribed by basic User Stories is not standardized across the development industry.

The basic hierarchy that is found in most *Agile* Scrum Product Backlogs follows the format of Epic → Feature

→ User Story → Task. The relationship between a User Story and a Task is a one-to-many relationship. Epics are the highest form of User Story representing an idea that is too large to comprehend by itself, or more clearly, "a large User Story that cannot be delivered as defined within a single iteration or is large enough to be split into smaller User Stories" (Cohn, 2004; Rehkopf, 2010). Many teams use Epics to track unrealized ideas within a Product Backlog prior to a team identifying the deliverable needed (Agile Alliance, n.d.). Features are the next step down from Epics. The relationship between Epics and Features is a one-to-many relationship (Britsch, 2017). Feature, though not well defined, are generally realized as a larger grouping of User Stories too big to be depicted as a single item, but too small to be considered Epics. User Stories clustered in this way are tied to specific common functionality (Britsch, 2017; Guay, 2019; Lee et al., 2003). The User Story is beneath Features in the hierarchy. The relationship between Features and User Stories is a one-to-many relationship. As described above, User Stories are an "end goal, not a feature, expressed from the software user's perspective" that is to be realized in the software system (Rehkopf, n.d.). Each User Story should be small enough to be completed within one Sprint cycle (Agile Modeling, n.d.; Britsch, 2017). In the most basic of terms, a User Story is "something a user wants" (Cohn, n.d.). Finally, User Stories can be further broken down into Tasks (Agile Modeling, n.d.). This hierarchical traceability is shown in Figure 5.

For example, *Agile* Scrum User Stories track traceability to roles through natural language in the Connextra format. Many times, this is expressed as a defined "Persona" (Maloney, n.d.; Szabo, 2017). The purpose of the persona is to assist the *Agile* Scrum team in discovering the value of the product they are making (Maloney, n.d.). A Persona is "a customer profile that puts a face on the facts by encapsulating and representing the data" (Broschinsky and Baker, 2008). In other words, Personas are a fictionalized representation of a customer type that *Agile* Scrum software developers use to gain insight into customer needs, goals, and requirements (Cooper, 2004; Szabo, 2017). This traceability can be useful but is limited as it does not identify specific stakeholders as requirement owners. Rather, it identifies customer types, which software developers may or may not have access to for elaboration of needs, goals, and requirements. *Agile* Scrum teams that do not use Personas often do not have firm definitions of user roles.

Precluding organizational directives, each *Agile* Scrum team is free to cherry-pick what parts of User Story requirements traceability the team desires. Some Scrum Teams will ensure that there is as high a level of traceability as possible for their projects. Other teams will ignore traceability altogether, outside of the minimal traceability that User Stories have built in inherently. Therefore, there cannot be a definitive case set forth that the majority of *Agile* Scrum teams utilize good requirements traceability.

Predictive-Style Requirements Management Traceability

INCOSE emphasizes that traceability is "critical to managing relationships" (INCOSE Requirements Working Group, 2023) and establishing traceability is an essential activity in the Technical Processes (INCOSE, 2023). In the INCOSE standard, traceability allows systems engineers to "understand then identify, location, relationships, pedigree, origin of data, materials, and parts of the objects/entities/items" (INCOSE, 2023).

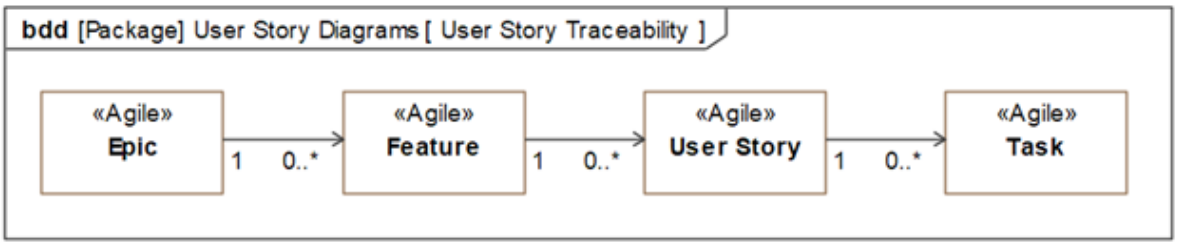

Figure 5. *Agile* Scrum User Story Hierarchy

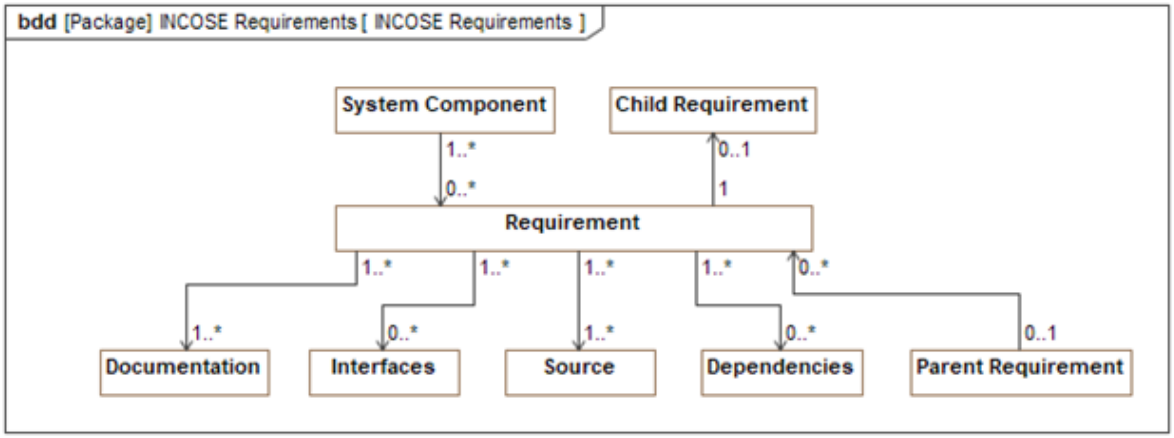

Figure 6. INCOSE Requirements Management Traceability

Requirements, both as a set and individually, are traced vertically and horizontally (INCOSE Requirements Working Group, 2023). Vertical traceability is of a hierarchical nature (INCOSE, 2023). Practitioners commonly refer to relationships of this type as "parent-child" relationships. The parent is directly "above" the child in the hierarchy. Tracing requirements in this manner allows for a top-down perspective of a system and defines a system's hierarchical architecture.

Horizontal traceability is used to trace requirement data through system lifecycles and across a given level of system architecture (INCOSE, 2023; INCOSE Requirements Working Group, 2023). Linkages of items generated in a lifecycle stage that are to be used by another lifecycle stage are an example of horizontal traceability. Horizontal traceability relationships are referred to as "peer" relationships (INCOSE, 2023). By tracing relationship connections both vertically and horizontally, a network representing the full view of the system of interest emerges, decreasing the effect of the large and interconnected systems' complexity.

Due to the way that requirements are documented in *Predictive* requirements management, any item can be traced to any other item. Documentation of linkages to other artifacts that each requirement traces to is part of a requirements register, linking each unique line item to other unique line items, as shown in Figure 6. Thorough tracing of requirements is not only encouraged but required in the technical processes (INCOSE, 2023). Without the built-in traceability, many requirements registers would be flat files and indecipherable to most people. Examples of common traceability found in INCOSE requirements are parent-child relationships, source derivation, interfaces, and dependencies (INCOSE, 2023).

Model-Based Systems Engineering (MBSE) activities expand upon traceability in *Predictive* requirements management, directly increasing the level of visibility and understanding that requirements offer. As MBSE efforts, specifically SysML, rely heavily on INCOSE requirements definition and standards, MBSE should be considered a part of *Predictive* Requirements Management rather than a neutral third-party methodology. Requirements managed in SysML allow for additional traceability relationships realized through metadata connections. These relationships include, but are not limited to, derive, refine, satisfy, verify, and trace (dos Santos Soares and Vrancken, 2008; Friedenthal et al., 2015; Wheaton and Herber, 2024). When making use of other MBSE tools such as Model-based Structured Requirements (MBSR), even more detailed traceability is allowed for, such that attributes "can be linked to other model elements that represent aspects of the physical system, precise mathematical conditions, test cases, etc." through the use of stereotypes (Herber and Eftekhari-Shahroudi, 2023). A stereotype is a form of metadata that defines how an existing meta class may be extended, enabling the use of platform, domain-specific terminology, or a notion in place of, or in addition to, the ones used for the extended meta class (Herber and Eftekhari-Shahroudi, 2023). Further, documenting

the rationale of requirements is incredibly important to traceability and the understanding of the "why" of the requirement itself. An MBSE engineer can easily accomplish this through the «rationale» stereotype in MBSE linked to a textual comment detailing the rationale (Herber, Narsinghani, and Eftekhari-Shahroudi, 2022). With modeling, linkage relationships are easily understood by a larger audience, as the media presented does not require mental modeling to visualize. Rather, the connections are plainly drawn in a graphical manner. The INCOSE methodology of integrating traceability directly into requirements management in an open and flexible manner is much more robust and useful for system modeling and understanding complex systems. INCOSE requirements traceability allows for modularity through the flexibility to allow a requirement to be traced to any system element, giving clear visibility into how requirements are being met. This visibility directly results in increased verification and validation capabilities through a clearer understanding of the system. Utilizing MBSE techniques, requirements can be traced directly to system architecture, better displaying how requirements are being realized through implementation. Tracing requirements back through derivations, sources, interfaces, documentation, and many other inputs allows systems engineers to fully conceptualize the problem space, building a mental model of not only what needs to be built, but also how and why each requirement is necessary.

Requirements Traceability Conclusions

Both *Agile* Scrum User Stories and INCOSE style requirements management have a form of traceability inherent in the system. The traceability is present in diverse ways in both systems. INCOSE requirements management uses a structured, bidirectional, and hierarchical methodology that has firm rules that requirements engineers must follow (INCOSE, 2023; INCOSE Requirements Working Group, 2023). *Agile* Scrum User Stories allow for a more diverse inheritance process, but it is generally considered a good form to follow a top-down hierarchical structure. Requirements management style "shall" statements have inherent traceability to any other item that may affect them, whereas *Agile* Scrum User Stories are self-contained, except in cases where teams use metadata, which is not generally proscribeda. Both styles of requirements have traceability, though INCOSE requirements management presents a more robust capability. Although both *Agile* Scrum User Stories and INCOSE requirements management methodologies include built-in traceability, User Stories' traceability is not as robust as requirements management style "shall" statements. User Stories have minimal traceability, prescribing only a "parent-child" relationship. This restriction can be overcome, but requires outside intervention using such tools as Jira. INCOSE requirements management builds modular and flexible traceability into the system, allowing for traceability to any other requirement or system element. Not only does it allow for a greater range of traceability capability, but requirements management also dictates that all requirements should name types of traceability outside of the "parent-child" relationship, such as requirement derivation, verification methods, how requirements are refined, and how requirements are satisfied in the system. This allows for a more robust mental model to be built by the development team, facilitating an increased likelihood of delivering what the customer needs while minimizing rework that results in project cost, schedule, and performance slippages. It is clear that although User Stories do have minimal traceability inherent in the Connextra format, INCOSE requirements management outperforms User Stories in every metric.

Goals versus Requirements

Introduction to Goals

ISO/IEC/IEEE 29148-2018 states the following about goals, "The term 'Goal' (sometimes called 'business concern' or 'critical success factor') refers to the overall, high-level objectives of the system. Goals provide the motivation for a system, but are often vaguely formulated. It is important to assess the value (relative to priority) and cost of goals" (IEEE, 2011). Goals are abstract expressions of a customer's desire for a system function or performance, whereas requirements often describe a functionality at a much lower level. Goals

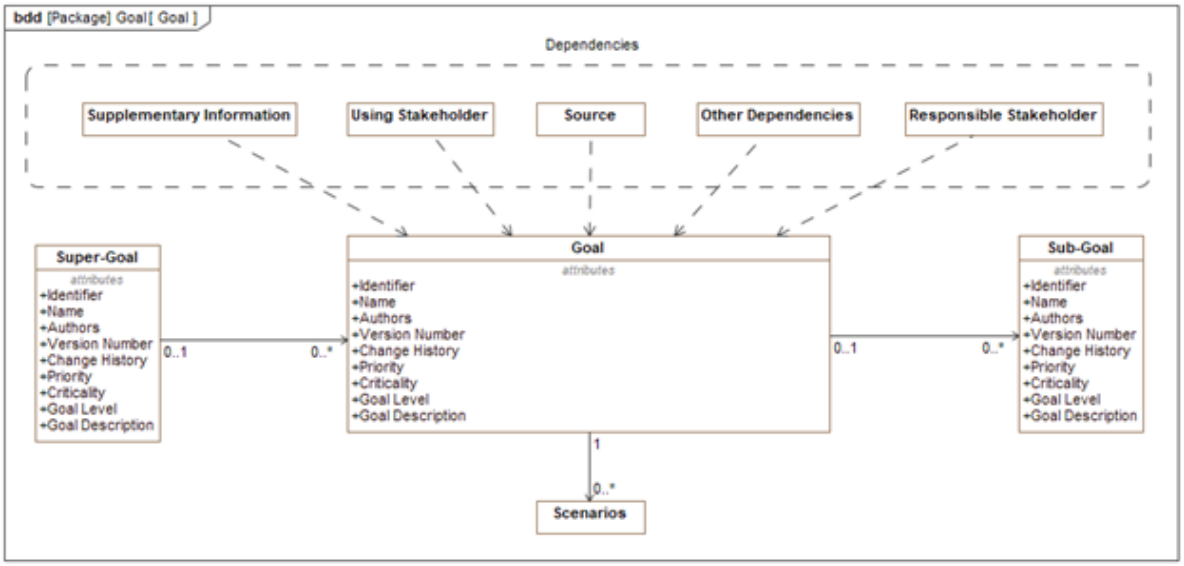

Figure 7. Goals - Traceability and Relationships

are normally vaguely formed and are tied to cost-to-benefit analysis and document stakeholder intention, refining the system vision into objectives to be fulfilled (Pohl, 2010).

Goals are commonly associated with scenarios in requirements management (Pohl, 2010). Pohl states scenarios "describe concrete system interactions and hence enable the stakeholders to describe concrete examples of satisfying the identified goals" (Pohl, 2010). Goals describe what the stakeholders want. Scenarios describe situations in which the goal is used or needed. Goals and scenarios work well together to assist in the development of new requirements as stakeholders detail what the system "should" do (Pohl, 2010). Goal traceability and relationships are shown in Figure 7.

User Stories as Goals

User Stories are not detailed enough to be considered true requirements, as they do not have all the metadata associated with an INCOSE requirements management style requirement. The simple structure of the Connextra User Story format prevents a full understanding of the domain and system function and relations to be drawn from a User Story (Günes and Aydemir, 2020). User Stories are realistically goals and not requirements. In the Connextra format, the word "goal" is used instead of requirement, further reinforcing this (Lucassen et al., 2016). As User Stories are more aligned with goals because they lack the detail to be considered true requirements. Thus, documentation using solely ambiguous User Stories creates problems in understanding the proposed system domain and functionality. The problems, if left undetected, propagate throughout the *Agile* software development process and documentation, which leads directly to a result of negative effects on the system in development (Amna and Poels, 2022). By relying on goals instead of detailing requirements at the lowest level possible, software developers risk incomplete understanding of the problem domain resulting in improper software architecture. This outcome leads to waste in the form of rework with impacts on cost, schedule, and performance in a project (Boehm and Turner, 2004).

Conclusions

Predictive-style requirements offer more engineering and technical rigor through built-in robust traceability and metadata than User Stories, however, User Stories offer a high-level, abstract, and easy-to-execute system. Both forms of requirements management offer minimal traceability to user, or persona, types, needed function, and reasoning. User Stories are used in *Agile* software development, and *Predictive*-style requirements are more often used in traditional, *Predictive* projects. Neither lends themselves solely to fully successful software project execution.

Agile Scrum User Stories should not be considered requirements when compared to the *Predictive* definition, as User Stories are too abstract to communicate customer needs. Instead, they are goals. They are abstract definitions of what a customer needs and wants in a system. The abstract nature of User Stories can directly cause the introduction of defects, both in requirements and in the system, through misunderstanding or miscommunication with the customer. This lack of understanding of the problem domain causes difficulty in the performance of thorough Verification and Validation activities, increasing development cost and adversely affecting the project schedule. Without a detailed baseline of complete and correct requirements, the introduction of technical debt occurs in the form of poor documentation and a lack of overall organizational knowledge of system structure, which in turn impedes the knowledge transfer of architecture and functional understanding.

Predictive-style "shall" statements are concrete statements of what a customer needs a system to do. These statements can be too rigid in management and tracking to allow for the flexibility software developers need to perform *Agile* development. By requiring in-depth, detailed, and atomic requirements up front, the flexibility inherent in software development is lost. Writing overly detailed requirements becomes a wasted effort because many requirements become obsolete as customer feedback is incorporated into work backlogs. A complete and consistent set of requirements is necessary for full system Verification and Validation activities. *Predictive*-style requirements management provides this capability despite the significant drawback of reduced flexibility in project performance.

The traceability of requirements is imperative for understanding the full system domain. Requirement traceability is inherent in *Predictive*-style requirements management, as not only are parent-child relationships capable down to the n^{th} level, but MBSE and other modern *Predictive*-style requirements management methodologies allow for full traceability to all system elements. In contrast to *Predictive*-style requirements management, *Agile* requirements management does not have thorough traceability horizontally and vertically throughout the system life cycle. This creates difficulties in the performance of Verification and Validation activities, possibly introducing defects, lack of documentation, and impaired understanding of the problem domain. *Predictive*-style requirements management has a very thorough and rigorous traceability inherent in the system, but the drawbacks hinder the acceptance of this method in software projects.

Both *Predictive*-style "shall" statements and User Stories have benefits and drawbacks when applied to *Agile* software projects, and neither individually fully captures the needs of development teams with regard to executing projects. Due to the lack of large-scale traceability and metadata linkages, *Agile* User Stories alone do not meet the needs of critical software systems, such as systems in the defense, medical, aerospace, and financial sectors. *Predictive* requirements management is not flexible on its own to allow ease of tracking in *Agile* Scrum Sprints, as requirements are considered too "low" in scope. On the basis of the above reflections on the current state of *Agile* and *Predictive* requirements management, it is clear that a dual system is needed. Initial requirements elicitation through *Predictive* requirements management methods should be performed to create a requirements baseline. From this initial baseline, Epics can be drawn and traced directly to individual requirements. In this way, the flexibility of User Stories can be taken advantage of while the traceability and rigorous technical documentation of *Predictive* requirements management can still be employed.

References

Agile Alliance. (n.d.). Agile glossary [Accessed 14 APR 2024]. https://www.agilealliance.org/glossary

Agile Modeling. (n.d.). User stories: An agile introduction [Accessed 14 APR 2024]. https://agilemodeling.com/artifacts/userStory.htm

Amna, A. R., & Poels, G. (2022). Ambiguity in user stories: A systematic literature review. *Information and Software Technology*, *145*.

Ashmore, S., & Runyan, K. (2014). *Introduction to agile methods*. Addison-Wesley Professional.

Atlassian. (2023). R4j - requirements management for jira [Accessed 27 JAN 2024]. https:////marketplace.atlassian.com/apps/1213064/r4j-requirements-management-for-jira?tab=overview&hosting=cloud

Azanha, A., Argoud, A. R. T. T., d. Camargo Junior P D, J. B., & Antoniolli, P. D. (2017). Agile project management with scrum. *International Journal of Managing Projects in Business*, *10*(1), 121–142.

Bijan, Y., Yu, J., Stracener, J., & Woods, T. (2013). Systems requirements engineering—state of the methodology. *Systems Engineering*, *16*(3), 251–377.

Boehm, B., Lane, J. A., Koolmanojwong, S., & Turner, R. (2007). Agile software development. Springer.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley/Pearson Education.

Britsch. (2017). The basics: Epics, stories, themes & features," the digital business analyst [Accessed 14 APR 2024]. https://thedigitalbusinessanalyst.co.uk/epics-stories-themes-and-features-4637712cff5c

Broschinsky, D., & Baker, L. (2008). Using persona with xp at landesk software, an avocent company. *Agile 2008 Conference*.

Chief Information Officer. (2010). Dodaf dod architectural framework version 2.02 [Accessed 03 MAR 2024]. https://dodcio.defense.gov/library/dod-architecture-framework/

Cohn, M. (n.d.). Epics, features and user stories [Accessed 14 APR 2024]. https://www.mountaingoatsoftware.com/blog/stories-epics-and-themes

Cohn, M. (2004). User stories applied for agile software development. *XP Atlanta*.

Cooper, A. (2004). *Inmates are running the asylum, the: Why high-tech products drive us crazy and how to restore the sanity*. Sams Publishing.

Das, S., Ali, Z., Bandi, S.-N., Bhagat, A., Chandrakumar, N., Kucheria, P., Pariente, M., Singh, A., & Tipping, B. (2021). Engineering artificially intelligent systems. Springer.

Davenport, T. H. (2005). *Thinking for a living: How to get better performances and results from knowledge workers*. Harvard Business Review Press.

Dingsoyr, T., Dyba, T., & Moe, N. B. (2010). *Agile software development: Current research and future directions*. Springer Berlin, Heidelberg.

dos Santos Soares, M., & Vrancken, J. (2008). Model-driven user requirements specification using sysml. *Journal of Software*, *3*(6), 57–68.

Dove, R., Lunney, K., Orosz, M., & Yokell, M. (2023). Systems engineering agility in a nutshell. *INCOSE Insight*, *26*(2).

Flora, H. K., & Chande, S. V. (2014). A systematic study on agile software development methodologies and practices. *International Journal of Computer Science and Information Technologies*, *5*(3), 3626–3637.

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, *9*, 28–35.

Friedenthal, S., Moore, A., & Steiner, R. (2015). *A practical guide to sysml: The systems modeling language*. Elsevier Inc.

Grenning, J. (2017). Agile uprising podcast: Interview with james grenning.

Guay, C. (2019). Scrum tips: Differences between epics, stories, themes and features [Accessed 14 APR 2024]. https://const.fr/blog/agile/scrum-differences-epics-stories-themes-features/

Günes, T., & Aydemir, F. B. (2020). Automated goal model extraction from user stories using nlp. *2020 IEEE 28th International Requirements Engineering Conference (RE)*.

Herber, D. R., & Eftekhari-Shahroudi, K. (2023). Building a requirements digital thread from concept to testing using model-based structured requirements applied to thrust reverser actuation system development. *9th International Conference on Recent Advances in Aerospace Actuation Systems and Components*.

Herber, D. R., Narsinghani, J. B., & Eftekhari-Shahroudi, K. (2022). Model-based structured requirements in sysml. *IEEE 2022 International Systems Conference (SysCon)*.

Hines, P., Holweg, M., & Rich, N. (2004). Learning to evolve: A review of contemporary lean thinking. *International Journal of Operations & Production Management*, *24*(10), 994–1011.

Hohl, P., Klünder, J., van Bennekum, A., Lockard, R., Gifford, J., Münch, J., Stupperich, M., & Schneider, K. (2018). Back to the future: Origins and directions of the "agile manifesto" – views of the originators. *Journal of Software Engineering Research and Development*, *6*(15).

IEEE. (2011). *Systems and software engineering — life cycle processes — requirements engineering*.

INCOSE. (2023). *Systems engineering handbook, 5th edition*.

INCOSE Requirements Working Group. (2023). *Guide to writing requirements*. International Council on Systems Engineering (INCOSE).

Jorgensen, M. (2004). A review of studies on expert estimation of software development effort. *The Journal of Systems and Software*, *70*(2), 37–60.

Lee, C., Luigi, G., & Xiaoping, J. (2003). An agile approach to capturing requirements and traceability. *Proceedings of the 2nd international workshop on traceability in emerging forms of software engineering (TEFSE 2003)*.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E. M., & Brinkkemper, S. (2016). The use and effectiveness of user stories. *Requirements Engineering: Foundation for Software Quality*.

Mahnic, V., & Hovelja, T. (2012). On using planning poker for estimating user stories. *The Journal of Systems and Software*, *85*(9), 2086–2095.

Maloney, B. (n.d.). What are user personas? [Accessed 14 APR 2024]. https://resources.scrumalliance.org/Article/user-personas

Mountain Goat Software. (2024). Planning poker [Accessed 27 JAN 2024]. https://www.mountaingoatsoftware.com/agile/planning-poker

Paulk, M. C. (2013). A scrum adoption survey. *Software Quality Professional*, *15*(2).

Pohl, K. (2010). *Requirements engineering: Fundamentals, principles, and techniques*. Springer Berlin, Heidelberg.

Project Management Institute. (2017). *Agile practice guide*.

Project Management Institute. (2021). *The standard for project management and a guide to the project management body of knowledge*.

Rauf, A., & Al Ghafees, M. (2015). Gap analysis between state of practice & state of art practices in agile software development. *Proceedings of Agile Conference (AGILE)*.

Rehkopf, M. (n.d.). User stories with examples and a template [Accessed 14 APR 2024]. https://www.atlassian.com/agile/project-management/user-stories#:~:text=A%20user%20story%20is%20the,the%20end%20user%20or%20customer

Rehkopf, M. (2010). Agile epics: Definition, examples, and templates [Accessed 03 MAR 2024]. https://www.atlassian.com/agile/project-management/epics.%20%5BAccessed%2014%2004%202024%5D/

Rosson, D. (2024). *Merging systems engineering methodologies with the agile scrum framework for department of defense software projects* [Doctoral dissertation, Colorado State University].

Schwaber, K., & Sutherland, J. (2020). *The scrum guide*. Ken Schwaber; Jeff Sutherland.

Szabo, P. W. (2017). *User experience mapping*. Packt Publishing.

Verwijs, C. (2020). Thinking by sprinting: What cognitive science tells us about why scrum works [Accessed 25 JAN 2024]. https://www.scrum.org/resources/blog/thinking-sprinting-what-cognitive-science-tells-us-about-why-scrum-works

Verwijs, C., & Russo, D. (2023). A theory of scrum team effectiveness. *ACM Transactions on Software Engineering and Methodology*, *32*(3), 1–51.

Wheaton, J., & Herber, D. R. (2024). Digital requirements engineering with an incose-derived sysml meta-model. *Conference on Systems Engineering Research (CSER) 2024*.

Wysocki, R. K. (2012). *Effective project management: Traditional, agile, extreme, sixth edition*. Wiley.