A Conceptual Model for Integrating Agile Practices into Systems Engineering Management

Ottaviani F.M.*, Zenezini G.*, Rebuglio M.*, Narbaev, T.*, De Marco, A.*

** Dipartimento di Ingegneria Gestionale e della Produzione Industriale, Politecnico di Torino, Corso Duca degli Abruzzi, 24 10129 – Torino – Italy (filippo.ottaviani@polito.it, giovanni.zenezini@polito.it, massimo.rebugli@polito.it, timur.narbaev@polito.it, alberto.demarco@polito.it)*

Abstract: Current technological advancements and escalating complexity in Systems Engineering (SE) are driving systems shift from pre-specified to evolutionary life cycle models. While Agile principles align with this shift, many SE projects are subject to technological and management constraints that limit the implementation of Agile principles. Existing literature does not provide standard integration models in dependencies- and regulatory restrictions-intensive systems. This study provides a conceptual model for integrating Agile practices into SE projects. The model, defined using the Unified Modeling Language (UML) class diagram notation, encompasses SE and Agile Project Management (PM) classes along with their attributes, methods, and relationships. The model provides operation-level support, including requirements engineering, dependency modeling, concurrent features development, progress monitoring, and quality management. Beyond serving as a roadmap for managing Agile SE projects, the model lays the groundwork for developing dedicated Agile SE management applications. The proposed model can enhance Agile SE management practices, increasing the chances of successful SE projects.

Keywords: Systems Engineering, Project Management, Agile Development, Conceptual Modeling

1. Introduction

Systems engineering (SE) requires equal focus on management and technical expertise. From the customer's perspective, the value of a system depends on technical factors and economic factors, including the cost associated with the system life cycle (LC) (Blanchard, 2004). Consequently, projects must deliver the intended results while meeting cost and time constraints.

Rapid technological advances are driving major changes in SE, increasing its complexity. In response, Agile SE has emerged as a solution to address this need for resilience. The approach adapts agile principles to SE, emphasizing architectures that enable structural and functional changes, regardless of whether the focus is on the development process or the final product (Dove & LaBarge, 2014b).

For SE to be Agile, it must satisfy two conditions: resolving technical uncertainties before system release (Haberfellner & de Weck, 2005) and adapting management processes in parallel (Bonnet et al., 2015). This adaptation involves aligning business objectives with agile principles, empowering teams and individuals to make decisions and embrace change, implementing flexible, responsive workflows, and utilizing tools and infrastructure that promote agility (Ebert & Kirschke-Biller, 2021).

While Agile SE offers significant potential benefits, its implementation may not be straightforward. Agile approaches in SE extend across the entire system LC, impacting all organizational aspects beyond just the definition and development phases. For success, this comprehensive approach necessitates LC management methods and tools that align with Agile principles. Specifically, requirements engineering, modeling, traceability, verification, and continuous build and deployment.

In addition, constraints can hinder the adoption of agile practices. Contractual obligations and task dependencies can limit flexibility and make it difficult to respond quickly to evolving requirements. Successful Agile SE implementation therefore depends on understanding how to adapt to rapid and frequent scope changes throughout a project. This also requires transforming traditional management processes (i.e., planning, monitoring, controlling, and risk management) to align with agile principles.

Scrum is a widely used framework for Agile SE (Dingsøyr et al., 2012). The framework aligns with evolutionary, sequential system LC models. However, it is essential to note that Scrum's focus on process does not directly address the infrastructure process management, which is essential for maintaining agility within the process architecture. On the product side, Scrum's roots in software development present certain limitations. Traditional software development leverages Object-Oriented Programming (OOP), which inherently promotes modularity and flexibility, aiding system adaptability (Dove & LaBarge, 2014a).

Implementing Agile methodologies in complex systems requires a surrogate model that can be rapidly modified

and serves as a continuous verification tool to facilitate Agile processes (Bott & Mesmer, 2020). Evaluation may lead to adopting an incremental and iterative development approach for complex systems with evolving requirements. This incremental development method is enabled using an open system architecture like Model-Based Systems Engineering (MBSE). Models ensure consistency between system and software requirements, enabling seamless progression into the design and validation stages (Ebert & Kirschke-Biller, 2021).

Research focused on Agile SE remains relatively limited compared to other areas within SE. Many existing studies concentrate solely on Agile SE or Agile Project Management (PM), without adequately bridging these disciplines' gaps. While frameworks like ASELCM (Dove, 2017; Dove & Schindel, 2019; Hause, 2023; Schindel & Dove, 2016) seek to integrate Agile SE with Scrum and MBSE, they often operate at a high level. This focus on the entire system LC often neglects the execution phase.

This study aims to shed light on the practical implementation of Agile SE practices. To achieve this, the study provides a conceptual model as a class diagram. This diagram outlines the structure and relationships between the classes derived from Agile SE and PM. The model allows just-in-time baselining of system requirements, dependency modeling, performance monitoring, and quality management.

This paper begins with an introduction outlining the background, identified theoretical gaps, and the study's primary objective. It proceeds with a review of relevant literature within the field. The methodology section describes the development of the proposed conceptual model. The discussion section analyzes the theoretical and practical contributions derived from the study and acknowledges the model's limitations. Finally, the paper concludes by outlining the general limitations of the research and suggesting directions for future investigations.

2. Literature Review

2.1 Agile Development and Systems Engineering

SE literature defines *agile systems* as those capable of thriving under unpredictable, uncertain, and changing conditions (INCOSE, 2015). These systems can reconfigure goals, requirements, plans, assets, and products. They also empower individuals (engineers, designers, customers) as "product owners" (Forsberg, Kevin and Mooz, Hal and Cotterman, 2005). These definitions align with the Agile Manifesto for Software Development (Martin Fowler & Highsmith, 2001), which proposed PM principles advocating a shift away from the waterfall model.

To effectively respond to uncertainty, Agile SE must incorporate non-traditional development approaches. Unsurprisingly, Agile SE and Agile PM share common traits: Agile SE flexibly adapts to shifting demands and technical requirements. At the same time, Agile PM rapidly responds to new information arising during development (Haberfellner & de Weck, 2005).

As previously mentioned, much of the literature focuses on either comparing Agile PM and SE, treating them as distinct approaches, or devising optimal Agile implementation strategies within SE (Darrin & Devereux, 2017). Haberfellner & de Weck (2005) classify this latter approach as "agile systems–engineering," contrasting it with "agile–system engineering," which focuses on embedding agility within the systems themselves. In this study, we argue that Agile PM can facilitate the development of agile systems.

Several handbooks and standards guide Agile PM applications. Scrum, developed by Jeff Sutherland, John Scumniotales, and Jeff McKennainin in 1993, is widely adopted due to its adaptability to hardware, full LC, small systems (Kanavouras et al., 2022). Extreme programming (XP) often complements Scrum to enhance effectiveness (Rahman et al., 2022). Frameworks like ASELCM position Scrum within a broader context (Schindel & Dove, 2016). Another notable Agile PM approach is SAFe®, created by Dean Leffingwell, a knowledge base of proven, integrated principles, practices, and competencies for achieving business agility using Lean, Agile, and DevOps.

However, most Agile PM methods lack explicit guidance on the active infrastructure process management needed to sustain process architecture agility (Dove, 2014). Some attempts to bridge this gap include Wolff et al. (2021), who proposed integrating BizDevOps and scenario methods into Agile PM frameworks, particularly SAFe®. Rosser et al. (2014) presented an Agile SE framework aligned with SAFe®'s iterative development and backlog management. Dove et al. (2018) and Dove (2018) explored adapting and applying SAFe® within Agile SE, using features and epics for hardware-firmware integration (Dove et al., 2018), and examining hybrid Scrum-SAFe® approaches (Dove, 2018).

Theoretically, agile development principles provide a framework for cross-functional collaboration between systems and software engineers in hardware and software projects (Marbach et al., 2015). Agile SE should flexibly select and adapt appropriate agile methods based on experience rather than rigidly adhering to complex process models (Ebert & Kirschke-Biller, 2021). Specifically, for systems with life-or-death components, SCRUM and XP methods may be preferable to SAFe® (Rahman et al., 2022). Agile PM practices benefit Agile SE by addressing how agile teams often overlook the increasing complexity and dependencies within a system as they focus on incremental implementation (Ebert & Kirschke-Biller, 2021).

2.2 Project Management Conceptual Models

Using conceptual models is not new to scientific literature. Methods like data flow diagrams, entity-relationship diagrams, and object-oriented modeling provide standard architectures for organizations of various sizes and industries.

Raimond (1987) and Björk (1992) demonstrated the value of conceptual data modeling for representing information within PM systems. This approach models the structure of project data, including products, processes, resources, and more. They highlighted the potential of conceptual modeling over traditional methods. Building upon this, Luiten et al. (1993) proposed standard models consisting of a data model, a domain model (for expressing project concepts), and a project model to store relevant project data.

Several studies focused on developing reference models for PM, addressing core functions like cost, time, scope, and quality management. Karim & Adeli (1999) and Fadillah & Fitriana (2019) employed object-oriented programming to create PM information models encompassing various PM classes. Yeganegi & Safaeian (2012) emphasized the importance of mapping stakeholder influence within the model. Ottaviani et al. (2023) proposed an extensive conceptual model for standardizing PM, serving as a foundation for building PM information system software and databases.

2.3 Summary

Table 1 provides a summary of the aforementioned studies. The Topic columns indicate whether the study addressed SE, Agile PM, or traditional PM. The Level column indicates whether the study contribution is at the conceptual, procedural, or operative level.

Table 1: References studies topics addressed and level

Study	Topic			Level
	SE	Agile PM	PM	
Raimond (1987)			✓	Operative
Björk (1992)			✓	Operative
Luiten et al. (1993)			✓	Operative
Karim & Adeli (1999)			✓	Conceptual
Haberfellner & de Weck (2005)	✓			Conceptual
Dingsøyr et al. (2012)		✓		-
Yeganegi & Safaeian (2012)			✓	Conceptual
Dove (2014)	✓	✓		Conceptual
Rosser et al. (2014)	✓	✓		Conceptual
Dove & LaBarge (2014a)	✓			Conceptual
Dove & LaBarge (2014b)	✓	✓		Operative
Marbach et al. (2015)	✓	✓		Operative
Schindel & Dove (2016)	✓	✓		Procedural
Darrin & Devereux (2017)	✓	✓		Procedural
Dove (2017)	✓	✓		Operative
Dove (2018)	✓	✓		Operative
Dove et al. (2018)	✓	✓		Operative
Dove & Schindel (2019)	✓	✓		Conceptual
Fadillah & Fitriana (2019)			✓	Conceptual
Bott & Mesmer (2020)	✓	✓		Conceptual
Wolff et al. (2021)	✓			-
Ebert & Kirschke-Biller (2021)	✓			Conceptual
Rahman et al. (2022)	✓	✓		-
Kanavouras et al. (2022)	✓	✓		Conceptual
Hause (2023)	✓	✓		Conceptual
Ottaviani et al. (2023)			✓	Operative
This study	✓	✓	✓	Operative

3. Methodology

3.1 Notation

This study presents a conceptual model of the Agile SE management approach utilizing the Unified Modeling Language (UML) class diagram notation. UML, as a widely used visual modeling language, provides a standardized framework for representing the design of a system, facilitating the visualization of its core components and their interactions. The diagram depicts the system's essential components, including classes, their attributes, methods, and relationships between them.

In UML class diagrams, classes serve as blueprints for objects, encapsulating both state (attributes) and behavior (operations). Attributes are defined by their corresponding data types. Solid lines illustrate associations between classes, with numerical indicators at the ends denoting cardinality (e.g., one-to-one, one-to-many). Solid lines terminating in a diamond shape indicate a composition relationship, where the existence of a child class is strictly dependent on its parent class. Arrows denote inheritance relationships, illustrating the specialization of a class from a more generic class.

3.2 Model

Figure 1 presents the proposed conceptual integration model. Color-coding distinguishes classes related to SE (blue), Agile PM (red), traditional PM (gray), and those

that bridge disciplines (purple). The physical data model emphasizes management objects. This focus streamlines the process, shortening requirements engineering processes, thus reducing complexity. Each project (*Project*) involves stakeholders (*Stakeholder*), which are distinguished by their role (*Role*) – customer, contractor, subcontractor, or supplier. Each stakeholder articulates needs (*Need*), which form the basis for system requirements (*Requirement*), as per ISO/IEC/IEEE 29148:2018 and SEBoK (INCOSE et al., 2023). Requirements are distinguished based on their type (*Type*) – functional, quality, or constraints.

Features (*Feature*) address requirements and are associated to specific elements (*Element*) of subsystems within the system of interest (*System*), as per ISO/IEC/IEEE 15288:2023 and ISO/IEC 26702:2007. Features are listed in the product backlog (*ProductBacklog*), and their priority is determined through the *udPriority()* method. Once completed, features are made available through releases (*Release*). We assume that releases take place at the planned date (*planDate*) due to contractual obligations, while features completed and made available can vary.

Feature development occurs through user stories (*UserStory*). Stories may precede or succeed other user stories. Stories are committed during an iteration and are prioritized within the iteration backlog (*IterationBacklog*) through the *udPriority()* method.

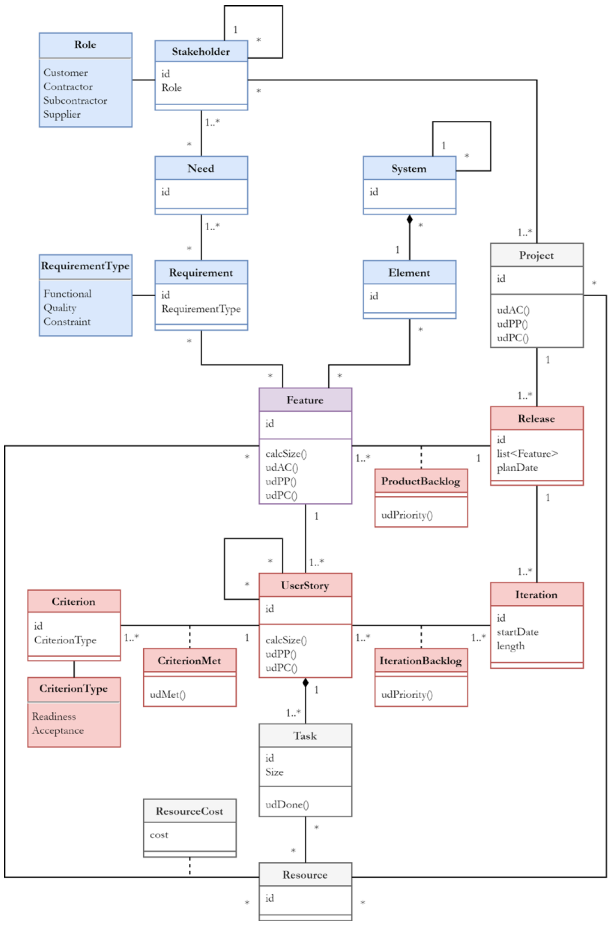

Figure 1: Agile SE and PM integration class diagram

Stories have associated criteria (*Criterion*), which type (*CriterionType*) is either *Readiness* or *Acceptance*. Readiness criteria determine whether a story can be added to the *IterationBacklog*, whereas *Acceptance* criteria determine whether it can be marked as done – *udDone()*. The *udMet()* method within the association class *CriterionMet* determines whether a criterion is met.

The cost of the resources employed (*ResourceCost*) depends on the feature, calculated through the *udAC()* method. At the project level, the project's actual cost is determined by summing the features' actual cost through the *udAC(),* inherited through the release objects.

4. Discussion

This paper presents a theoretical model for integrating Agile SE, Agile PM, and PM. The model focuses on the feature class, linking system requirements (SE) to projects (PM) and stories (Agile PM). Compared to previous studies, the model provides operational-level support following a simplified approach to SE and Agile SE practices. While Agile SE classes align with traditional SE practices, their dynamic connection to features through releases implies iterative development, implementation, verification, and validation in response to changing stakeholder needs. The Agile PM classes draw inspiration from Scrum and XP (release and iteration plans), focusing on quality management through readiness and acceptance criteria. Acceptance criteria ultimately determine user story completion.

The model provides a simple visual aid for understanding how the two domains of Agile SE and PM are connected. Both disciplines remain unchanged, except for omitting the "epic" concept to avoid redundancy. This decision reflects the discordant definitions of "epic" in literature: Agile SE studies view it as a collection of features. In contrast, Agile PM studies consider it a larger story. Additionally, the model establishes a fixed process for creating user stories directly linked to stakeholder personas. Finally, the model assumes features are atomic (without precedence relationships), while stories (not visible to the customer/client) can incorporate precedence constraints managed through prioritization in the iteration backlog.

The proposed model clarifies the logical architecture connecting SE, Agile PM, and PM components. It also lays the foundation for developing software tools to facilitate agile management within Agile SE projects. The model supports progress monitoring by tracking completed story points at the user story, feature, and project levels. Moreover, its transparency and simplicity allow easy integration into existing management applications.

This model, like any conceptual model, has inherent limitations. Conceptual models offer simplified representations of reality and cannot be fully exhaustive. Additional classes, relationships, or methods may be required for specific implementations. A potential gap exists between the model and the actual system; careful management is necessary to bridge this gap. Moreover, advanced functions like stakeholder management,

dynamic risk management, or prescriptive decision support tools may necessitate modifications or extensions to this model.

5. Conclusions

In SE, systems life-cycle models are shifting from prescriptive or sequential models to concurrent and adaptable approaches. To fully realize Agile SE, management of SLCM phases must also embrace agile principles. However, integrating these approaches can be complex, requiring tailored Agile practices for optimal fit within Agile SE. While the literature supports the feasibility of this integration through theoretical discussions and case studies, it lacks a clear and concise operational model. This study addresses this gap by proposing a conceptual integration model for Agile SE and PM. The model clarifies SE management components by defining classes with attributes and methods and their interactions.

This study acknowledges additional limitations beyond those inherent to the methodological approach. Firstly, the analyzed literature drove the modeling choices and aimed for maximum simplification. While capturing every possible configuration would contradict the goal of minimizing process complexity, the model's purpose extends beyond universality. Its primary objective is to highlight the operational relationships between Agile SE, Agile PM, and quality, risk, and monitoring management. When applied to real-world scenarios, the model serves as a customizable foundation for accurately reflecting the specific dynamics of the application context.

References

Björk, B. C. (1992). A unified approach for modelling construction information. *Building and Environment*, 27 (2), 173–194.

Blanchard, B. S. (2004). *System Engineering Management* (3rd ed.). John Wiley & Sons. Hoboken, NJ, USA.

Bonnet, S., Voirin, J., Normand, V., & Exertier, D. (2015). Implementing the MBSE Cultural Change: Organization, Coaching and Lessons Learned. *INCOSE International Symposium*, 25 (1), 508–523. Seattle, AW, USA.

Bott, M., & Mesmer, B. (2020). An Analysis of Theories Supporting Agile Scrum and the Use of Scrum in Systems Engineering. *Engineering Management Journal*, 32 (2), 76–85.

Darrin, M. A. G., & Devereux, W. S. (2017). The Agile Manifesto, Design Thinking and Systems Engineering. *2017 Annual IEEE International Systems Conference (SysCon)*, 1–5. Montreal, QC, Canada.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85 (6), 1213–1221.

Dove, R. (2014). Agile Systems-Engineering AND Agile-Systems Engineering. *INSIGHT*, 17 (2), 6–10.

Dove, R. (2017). On Defining Agile Systems Engineering. *INSIGHT*, 20 (3), 75–76.

Dove, R. (2018). Synergy: Agile Systems Engineering and Product Line Engineering at Rockwell Collins. *INSIGHT*, 21 (2), 43–46.

Dove, R., & LaBarge, R. (2014a). 8.4.1 Fundamentals of Agile Systems Engineering – Part 1. *INCOSE International Symposium*, 24 (1), 859–875.

Dove, R., & LaBarge, R. (2014b). 8.4.2 Fundamentals of Agile Systems Engineering – Part 2. *INCOSE International Symposium*, 24 (1), 876–892.

Dove, R., & Schindel, B. (2019). Agile Systems Engineering Life Cycle Model for Mixed Discipline Engineering. *INCOSE International Symposium*, 29 (1), 86–104.

Dove, R., Schindel, W. B., & Garlington, K. (2018). Case Study: Agile Systems Engineering at Lockheed Martin Aeronautics Integrated Fighter Group. *INCOSE International Symposium*, 28 (1), 303–320.

Ebert, C., & Kirschke-Biller, F. (2021). Agile Systems Engineering. *IEEE Software*, 38 (4), 7–15.

Fadillah, A. P., & Fitriana, D. (2019). Design of Project Data Management Information System. *IOP Conference Series: Materials Science and Engineering*, 662 (2).

Forsberg, Kevin and Mooz, Hal and Cotterman, H. (2005). *Visualizing project management: models and frameworks for mastering complex systems*. John Wiley \& Sons.

Haberfellner, R., & de Weck, O. (2005). 10.1.3 Agile SYSTEMS ENGINEERING versus AGILE SYSTEMS engineering. *INCOSE International Symposium*, 15 (1), 1449–1465.

Hause, M. (2023). Agile MBSE: Doing the Same Thing We Have Always Done, but in an Agile Way with Models. *INSIGHT*, 26 (2), 31–33.

INCOSE. (2015). *Systems Engineering Handbook* (4th ed.). John Wiley & Sons, Inc. Hoboken, NJ, USA.

INCOSE, IEEE Systems Council, & Systems Engineering Research Center. (2023). *Guide to the Systems Engineering Body of Knowledge (SEBoK)*.

Kanavouras, K., Hein, A. M., & Sachidanand, M. (2022). Agile Systems Engineering for sub-CubeSat scale spacecraft. *ArXiv*.

Karim, A., & Adeli, H. (1999). OO Information Model for Construction Project Management. *Journal of Construction Engineering and Management*, 125 (5), 361–367.

Luiten, G., Froese, T., Cooper, G., & Cad, R. J. (1993). An information reference model for architecture, engineering, and construction. *1st International Conference on the Management of Information Technology for Construction*, *August*, 1–10.

Marbach, P., Rosser, L., Osvalds, G., & Lempia, D. (2015). Principles for Agile Development. *INCOSE International Symposium*, 25 (1), 524–537.

Martin Fowler, & Highsmith, J. (2001). *The Agile Manifesto*.

Ottaviani, F. M., Rebuglio, M., & De Marco, A. (2023). Project Management Information System Data Model Development and Explanation. *Proceedings of the 13th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, 210–217.

Rahman, M., Islam, S., Fardous, R., Yesmin, L., & Nandi, D. (2022). Applying Scrum Development on Safety Critical Systems. *International Journal of Information Technology and Computer Science*, 14 (5), 44–57.

Raimond, L. (1987). Information systems design for project management: a data modeling approach. *Project Management Journal*, 18 (4), 94–99.

Rosser, L., Marbach, P., Osvalds, G., & Lempia, D. (2014). 7.4.2 Systems Engineering for Software Intensive Projects Using Agile Methods. *INCOSE International Symposium*, 24 (1), 729–744.

Schindel, B., & Dove, R. (2016). Introduction to the Agile Systems Engineering Life Cycle MBSE Pattern. *INCOSE International Symposium*, 26 (1), 725–742.

Wolff, C., Tendyra, P., & Wiecher, C. (2021). Agile Systems Engineering in Complex Scenarios. *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 1, 323–328. Cracow, Poland.

Yeganegi, K., & Safaeian, S. (2012). Design of Project Management Information Systems. *International Conference on Industrial Engineering and Operations Management*, 2545–2551.