software

MDPI

Article

An Agile Model-Based Software Engineering Approach Illustrated through the Development of a Health Technology System

Moe Huss *, Daniel R. Herber and John M. Borky

Department of Systems Engineering, Walter Scott, Jr. College of Engineering, Colorado State University, Fort Collins, CO 80523, USA
* Correspondence: moe.huss@colostate.edu

Abstract: Model-Based Software Engineering (MBSE) is an architecture-based software development approach. Agile, on the other hand, is a light system development approach that originated in software development. To bring together the benefits of both approaches, this article proposes an integrated Agile MBSE approach that adopts a specific instance of the Agile approach (i.e., Scrum) in combination with a specific instance of an MBSE approach (i.e., Model-Based System Architecture Process—"MBSAP") to create an Agile MBSE approach called the integrated *Scrum Model-Based System Architecture Process (sMBSAP)*. The proposed approach was validated through a pilot study that developed a health technology system over one year, successfully producing the desired software product. This work focuses on determining whether the proposed sMBSAP approach can deliver the desired *Product Increments* with the support of an MBSE process. The interaction of the *Product Development Team* with the MBSE tool, the generation of the system model, and the delivery of the *Product Increments* were observed. The preliminary results showed that the proposed approach contributed to achieving the desired system development outcomes and, at the same time, generated complete system architecture artifacts that would not have been developed if Agile had been used alone. Therefore, the main contribution of this research lies in introducing a practical and operational method for merging Agile and MBSE. In parallel, the results suggest that sMBSAP is a middle ground that is more aligned with federal and state regulations, as it addresses the technical debt concerns. Future work will analyze the results of a quasi-experiment on this approach focused on measuring system development performance through common metrics.

Keywords: software development; Model-Based Software Engineering (MBSE); Agile; Scrum; system architecture; modeling; systems engineering (SE)

Citation: Huss, M.; Herber, D.R.; Borky, J.M. An Agile Model-Based Software Engineering Approach Illustrated through the Development of a Health Technology System. *Software* **2023**, *2*, 234–257. https://doi.org/10.3390/software2020011

Academic Editors: Sanjay Misra, Robertas Damaševičius and Bharti Suri

Received: 2 March 2023
Revised: 28 March 2023
Accepted: 12 April 2023
Published: 17 April 2023

1. Introduction

Model-Based Software Engineering (MBSE) is well known for managing complexity during system development processes [1]. MBSE for information-intensive systems could be considered an attempt to have a knowledge hub for the software development lifecycle. Additionally, the definition of a system model requires the use of a formal, standardized language, such as Unified Modeling Language (UML) [2]. Leveraging individual intellectual capabilities in software engineering is one of the many opportunities that MBSE may provide. However, it also introduces additional technical and organizational challenges, which have impacted its wide adoption [3,4].

Agile is a software development approach emphasizing continuous product delivery by using short development cycles known as "sprints" [5]. Although there has been significant adoption in recent years, discussion about software development projects not meeting expectations has been increasing [6–8]. Given that there are still challenges that hinder MBSE and Agile from unlocking their full potential, Agile MBSE presented itself as a possible resolution [9–12]. Therefore, this study adopts a specific instance of the Agile approach (i.e., Scrum) in combination with a specific instance of an MBSE approach (i.e.,

MBSAP [13]) to create an Agile MBSE approach called the integrated *Scrum Model-Based System Architecture Process (sMBSAP)*.

Several researchers have advocated for the benefits of integrating Agile and MBSE [9,10,14]. Researchers have also seen the need for techniques and methods that support documentation in Agile environments [15]. A technique that makes documentation more easily writable, manageable, and updatable is needed [16,17]. This research contributes to the literature by explaining a practical approach to implementing Agile MBSE. The research also applies the proposed approach to a health technology system. The proposed sMBSAP attempts to narrow the gap between Agile practitioners and document-based/waterfall practitioners by simplifying and streamlining system documentation and making it easier to develop, manage, access, and share.

2. Background

2.1. Software Development Lifecycle (SDLC)

The Software Development Lifecycle (SDLC) is a process for building and maintaining software systems. Several researchers have explained that system development lifecycle models have been strongly influenced by software, and so the two terms "system" and "software" might be used interchangeably in terms of SDLC [18], especially since system development encompasses software system development [18–20]. Others have clarified that systems engineering, software systems engineering, and software engineering have different areas of focus [21].

Currently, two major SDLC categories are used when managing software systems projects: (1) traditional and (2) Agile. Several commonly used traditional systems development methods are shown in Figure 1. Traditional systems development methods have been historically associated with Document-Based Systems Engineering (DBSE) practices [22]. On the other hand, Agile systems development methods are the other category of SDLC that came to life to address some of the challenges associated with traditional systems development methods. The first generation of Agile approaches, sometimes called lightweight development methods [23], are also shown in Figure 1.

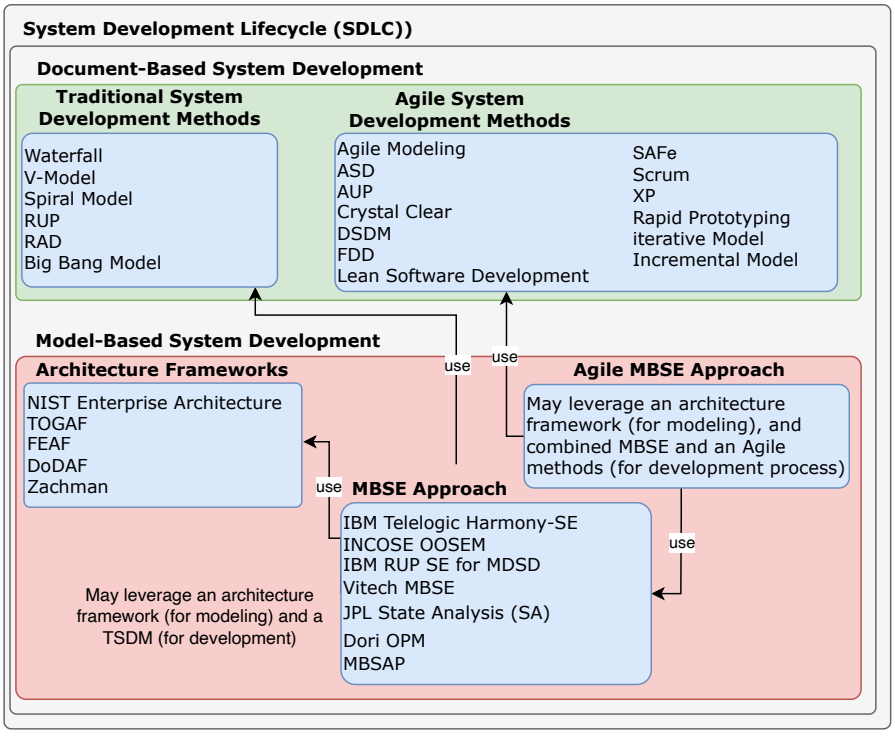

Figure 1. Overview of software development lifecycle models and their relationship with MBSE.

Although Agile methodologies have merits over traditional development methods, several limitations have been faced when expanding their adoption [24]. One of these limitations is that Agile methods considerably decrease the amount of required documentation [25], which is a source of subsequent issues. The lack of documentation does not align well with federal and state acquisition regulations. A report by the U.S. Government Accountability Office (GAO) identified 14 challenges related to implementing the Agile approach in federal agencies. These challenges include that government contracts may be designed with heavily structured tasks and performance checks that are not necessarily aligned with Agile methods or cadences [26]. A report [26] also highlighted the need to track several different Agile metrics, such as *Requirements* and architectures. Without tracking such metrics, the government may not have the right information for effective contract oversight [26]. The abilities of MBSE methodologies to centrally manage documentation and track *Requirements* and architectures make them a reasonable solution for aiding in complying with such regulations.

2.2. Scrum Method

The Scrum software development process is an Agile method for managing and directing the development of complex software and products by using incremental and iterative techniques [27]. The State of Agile Report revealed higher adoption of Scrum-based development in the present-day software industry compared to other methodologies. It was found that 87% of the 2022 survey participants leveraged Scrum [28].

The Scrum method includes three main components: roles, ceremonies, and artifacts [29]. The Scrum processes are grouped into five phases: initiate, plan and estimate, implement, review and retrospect, and release [30]. In addition, there are three distinct roles in the Scrum process: the *Product Owner*, the *Scrum Team*, and the *Scrum Master* [31]. The Scrum method includes periodic meetings known as ceremonies, which include *Sprint Planning*, *Daily Scrum (Standup)*, *Sprint Review*, and *Sprint Retrospective* [12,32–34]. In addition to the Scrum roles and ceremonies, the Scrum process delivers three main artifacts, namely, the *Product Backlog*, the *Sprint Backlog*, and the *Product Increment* [12,32–34]. A high-level overview of the Scrum ceremonies, artifacts, and phases is shown in Figure 2.

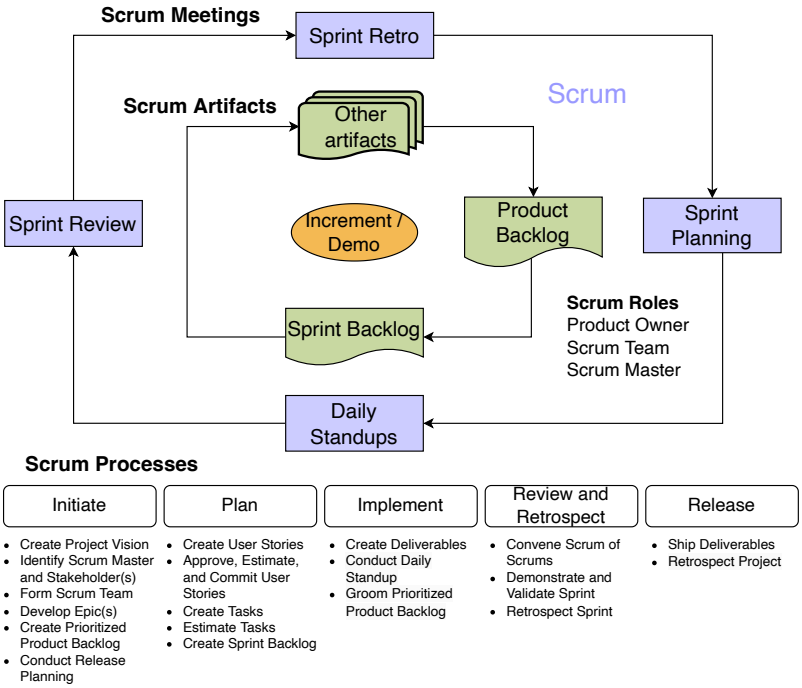

Figure 2. Scrum process overview.

Scrum artifacts are the key information that a Scrum team uses and generates during the sprints to show their progress to stakeholders. Some researchers even claimed that the code itself should act as a document [25]. Agile practitioners devalue comprehensive documentation and *traceability*; therefore, architecture artifacts, such as use case diagrams, data models, and *Requirement Traceability*, are not typical in Scrum methodology [14,31]. *User Stories* are used to capture scenarios or system *Behavior* by describing how a user interacts with the system [14]. With the Scrum approach, information elements are captured in an Agile tool (such as Jira [35] and ClickUp [36]) and amongst separate, independent documents by using Microsoft Office tools (rather than a primary, integrated system model as a single source of truth, as is the case in MBSE approaches).

2.3. Model-Based Software Engineering (MBSE)

Model-Based Software Engineering (MBSE) is a software development approach in which models play an important central role [37]. MBSE was developed to overcome the drawbacks of the conventional Document-Based Systems Engineering (DBSE) method, which became apparent as information-intensive systems became more complex [38]. The discussion around MBSE is closely tied to the concept of architecture, which represents the *Structure*, *Behavior*, and *Rules* for a complex entity (or system), including its evolution over time [13]. Some commonly encountered architecture frameworks are shown in Figure 1.

There has been a tendency to use MBSE approaches due to the reported benefits. The International Council on Systems Engineering (INCOSE) describes the benefits of an MBSE approach as improved communications, increased ability to manage system complexity, improved product quality, and enhanced knowledge capture [39]. However, it is important to note that none of the MBSE methodologies [13,40] shown in Figure 1 have gained the ever-increasing popularity of Scrum [28].

The concept behind model-based software engineering is to leverage software modeling to carry out development and maintenance and to achieve code reuse [3]. The notion of employing models to reduce software complexity has been around for many years. When appropriately used, MBSE can provide a significant opportunity to capitalize on individual intellectual assets in software engineering in general and to realize the promise of business/technology alignment made by Domain-Driven Design in particular [3]. However, MBSE can also pose a threat because of the additional challenges that it may introduce at the technical and organizational levels.

While adopting MBSE, several challenges have been identified and discussed by researchers and practitioners [38,41–43]. Some of these challenges include the disconnect between how system architects conceptualize their systems (within the limitations of a typical DBSE approach) and describe them in a different way. Another challenge is related to the perception that MBSE is performed by a tool, although several researchers explained that MBSE is more than just a tool [38,43,44]. Usability is another issue that has been described, as too many aspects and attributes have to be specified to describe a simple system characteristic—in other words, too many clicks [45]. Finally, the selection of MBSE tools without a deep understanding of user needs is another issue [45]. Accordingly, it is not a surprise that implementing an MBSE approach typically takes a long time and does not frequently provide incremental value to the customer. Therefore, searching for alternative approaches seems necessary.

Model-Based System Architecture Process (MBSAP)

The MBSAP outlines object-oriented design methods to create an architecture for a system through structured decomposition into modular and manageable levels of complexity by using object-oriented principles, such as abstraction, encapsulation, modularity, generalization, aggregation, interface definition, and polymorphism [13]. The process begins with identifying the customer's needs. Then, one iteratively develops progressive architecture models starting with an *Operational Viewpoint*, then a *Logical/Functional Viewpoint (LV)*, and,

finally, a *Physical Viewpoint*. Similarly, prototypes of the system (or system increments) are incrementally built, integrated, and tested with other increments. Finally, the cycle leads to either the delivery of a final product or a new starting point for a follow-on increment of development [13]. An overview of the MBSAP is shown in Figure 3.

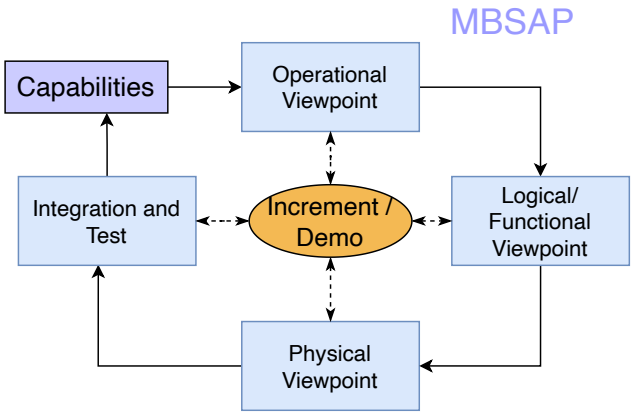

Figure 3. Overview of the MBSAP [13].

Many practitioners of object-oriented methods make the assumption that the essence of an object-oriented method is the incremental approach [13]. This incremental (spiral approach) originally evolved to respond to frequently changing *Requirements*, especially for complex systems. It is obvious that the MBSAP is intrinsically incremental and iterative (as shown by the closed loop in Figure 3), making its integration with other Agile methods occur naturally in an attempt to get the best of both approaches.

2.4. Agile MBSE

Agile MBSE presented itself as a possible solution for two issues that faced system development, namely, rigidity and waterfall orientation [9].

Agile MBSE also presented itself as a potential solution for the competing views and challenges related to documentation and *traceability*. The architecture specification document is usually very long, complex, and not self-explanatory [46]. Therefore, the Agile manifesto values "working software over comprehensive documentation" [47]. However, not all Agile practitioners seem to agree with this Agile principle [48]. Moreover, a survey conducted at the University of Melbourne revealed that despite the lower priority of documentation in Agile practices, 98% of the respondents considered documented information moderately to extremely important when estimating effort [49]. However, developers find documentation important, but at the same time, too little of it is available in their projects [48]. While some Agilistas devalue documentation and *traceability* [50], Agile methods and documentation are not actually contradictory [46,51]. A certain amount of documentation is essential [46,52]. Some researchers have also made a case that *traceability* is both necessary and required [14]. The view that supports documentation is adopted in government procurement/reporting practices [26]. Researchers see the need for techniques and methods that support documentation in Agile environments [15], such as a technique that makes documentation more easily writable, manageable, and updatable [16,17].

Douglass [14] clarified that in Agile MBSE, the outcome of Agile software development is implementation, while the outcome of systems engineering is specification. Douglass [14] also discussed the notion of model-based handoff to "downstream engineering", enabling precise and unambiguous communication between architecture and *Requirements* analysts and the discipline-specific teams.

Salehi and Wang [10] compared four V-models and found that they did not consider the Agile concept. This finding led to the proposal of the adoption of Agile in MBSE to create a new approach, which was termed the Munich Agile MBSE Concept (MAGIC) [10]. Integrating MBSE and product development offers the capability of building a virtual

prototype and a product's digital twin [11]. Bott and Mesmer [12] reviewed the theories supporting the Agile and MBSE methodologies and found them a key enabler of Agile methods for systems engineering. Figure 1 illustrates the relationship between SDLC models and MBSE.

3. Integrated Scrum Model-Based System Architecture Process (sMBSAP)

3.1. Research Methods

Context: The product development took place for one year (between May 2020 and May 2021) within an early-stage health technology startup company that aimed to develop a health tech system that provides dietary recommendations to users based on their health profiles. The development period coincided with the COVID-19 pandemic, when stay-at-home orders were in effect, so the *Product Development Team* primarily used virtual conferencing tools and Slack [53] for communication and collaboration.

Participants: The three co-founders, in addition to the lead author, served as the *Product Development Team*. The role of the lead author was then limited to developing the architectural artifacts based on requests from the team. Table 1 summarizes the product development team personas.

Table 1. Product development team personas.

Role	Gender	Industry Experience	Highest Education Level	Familiarity with System Development
System Architect and Scrum Master	Male	20+ years	MSc	Yes
Backend Developer	Male	15+ years	PhD	Yes
Frontend Developer	Male	15+ years	PhD	Yes
Health and Nutrition Scientist	Female	15+ years	PhD	No

Materials: Sparx Enterprise Architect ("Sparx EA") [54] was chosen as the architecting software tool, and Unified Modeling Language (UML) was chosen as the architecting language (which is allowed in MBSAP [13].) The sMBSAP artifacts generated included the *Product Backlog*, *Sprint Backlog*, *Glossary*, *Product Breakdown*, *Class Diagrams*, *Object Diagrams*, *Data Model*, *Activity Diagrams*, *Use Case Diagrams*, and *Capabilities* (*Requirements* and *User Stories*).

Procedures: Before each sMBSAP sprint, information elements were generated for each perspective, integrated into a system model by using Sparx EA, and referenced in the appropriate model viewpoint. The model elements, diagrams, and views were generated according to standard and non-standard UML diagram formats [13], and they were implemented according to the standard Sparx EA modeling procedures [54]. The sMBSAP procedures (outlined in the following sections) were implemented to conduct the sprints and create *Product Increments*.

The proposed sMBSAP approach followed a combination of Scrum and MBSAP, as shown in Figure 4. The processes near the bottom of the figure describe the steps followed by the *Product Development Team*, who used the sMBSAP to develop a software system; these processes will be described in detail in this section.

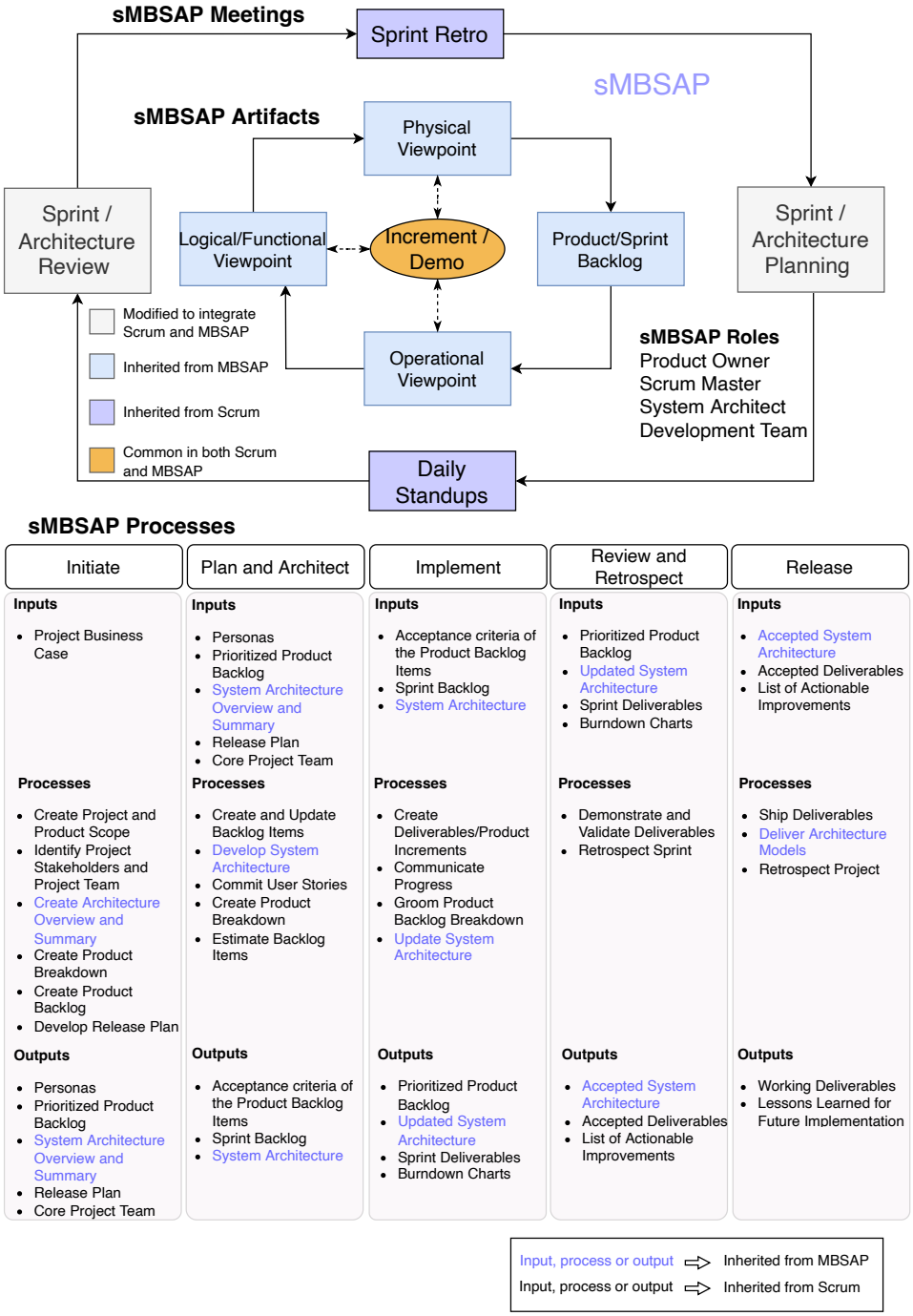

Figure 4. Overview of the sMBSAP with the goal of developing *Product Increments*.

3.2. Overview of the sMBSAP

Before describing the sMBSAP method in detail, it is important to highlight that the illustrative development activity shown here is for an implemented health technology system. The sMBSAP approach includes five phases: (1) Initiate, (2) Plan and Architect, (3) Implement, (4) Review and Retrospect, and (5) Release. The outputs of each phase serve as inputs to the following phase, as shown in Figure 4. Figure 4 also shows that two of the four main Scrum meetings are used during the sMBSAP approach, namely, *Daily Standups* and *Sprint Retro*. The other two Scrum meetings were modified for the sMBSAP. The sprint planning meeting was modified to be the *Sprint/Architecture Planning* meeting. The same applied to the *Sprint Review* meeting, which was modified to be the *Sprint/Architecture Review* meeting.

The typical MBSAP viewpoints generate the architecture artifacts for driving the development process. The MBSAP artifacts and *Sprint Backlog* that include *User Stories* are the key information that the *Product Development Team* uses to execute the product development and show the progress to stakeholders. The *Sprint Backlog* captures the list of items that need to be developed during each sMBSAP-driven sprint. The sMBSAP approach also includes the many typical MBSAP artifacts, including but not limited to a glossary, *Product Breakdown*, class diagrams, object diagrams, data models, use case diagrams, and capabilities.

According to Borky and Bradley [13], the term "capabilities" is a preferred term over the term "*Requirements*". *User Stories* are similar to *Requirements*, except they are written from the user's perspective—in other words, what a user shall do when using the system rather than describing what the system shall do for the user. The perspective of capabilities captures the system capabilities, which include *User Stories*, *Requirements*, and other behind-the-scenes tasks required to enable system capabilities. Now, the following description of the phases and processes of the sMBSAP is organized to mirror that of the Scrum method for more straightforward mapping.

3.3. Initiate

This phase includes the processes related to initiating the project. These processes are summarized below:

- Create Project and Product Scope: In this process, the project business case is reviewed to create a *Project Scope Statement* and subsequent *Product Scope*. The *Product Owner* is typically identified at this stage of the project.
- Identify Project Stakeholders and Project Team: In this process, the four project roles are identified, which include the *Product Owner*, *Scrum Master*, *System Architect*, and *Product Development Team*. Other project stakeholders are also identified during this process.
- Create Architecture Overview and Summary: In this process, an *Architecture Overview* that provides the following architecture-related information is created: the architecture's scope, purpose, and perspective, contextual information, the role of the *System Architect*, and the timeline of the architecture's development. The *Architecture Overview* acts as a contract between stakeholders and the *System Architect* based on establishing bilateral commitments and understanding of the role of the architecture effort within the overall project effort [13]. The main customer for the architecture artifacts is the *Product Development Team*; accordingly, the *System Architect* must explain the value and contribution of the architecture process. The *System Architect* should also expect organizational resistance and lack of support among software developers, especially those who are used to writing code with very little or no documentation as input. In this process, the *System Architect* decides which MBSE tool they will use throughout the project.
- Create Product Breakdown: In this process, the *Project Scope Statement* and *Product Scope* are used as the basis for breaking the product down into *Epics*, *Use Cases*, *User Stories*, and *Requirements*, as shown in Figure 5. The *Product Breakdown* is an iterative process that occurs first during the Initiate phase and is further refined through meetings between the project team and key stakeholders in subsequent phases as needed. In the sMBSAP approach, *Epics* are modeled as stereotyped *Use Cases* [14] and are decomposed to (lower-level) *Use Cases*, which are, in turn, decomposed into *User Stories*, which are broken down into *Requirements*. This taxonomy is shown in Figure 5.
- Create Product Backlog: In this process, *User Stories*, *Requirements*, and other tasks are added to the *Product Backlog*. These items are referred to as *Product Backlog Items (PBIs)*. It is important to note that a project team may choose to use only *User Stories* (commonly used in Scrum) or both *User Stories* and *Requirements* (*Requirements* are commonly used in MBSE and systems engineering). The *PBIs* will be progressively refined, elaborated, and later prioritized. The acceptance criteria are also established

at this point and will be further elaborated. The *Product Backlog* is developed and maintained by using a*Requirement* management tool or Agile development tool, such as Clickup [36]. Alternatively (or in addition), *User Stories* and *Requirements* may be visually captured in the model, as shown in Figure 6, which illustrates *User Stories* and *Requirements* that are modeled as stereotyped *Use Cases*, and *Requirements* are traced to *User Stories*. This allows a *User Story* to be described and to its connection to a persona to be shown. In this *Use Case* diagram, the *System Architect* wants to capture the interaction of the "User" with the "Health Assessment" module of the system and communicate it with the *Product Development Team*. The *System Architect* explains the "User" behavior through a combination of *User Stories* and *Requirements*. At the beginning of the "Health Assessment", the system displays a series of messages to the "User" to allow them to customize their "Health Assessment" experience. The "User" will specifically be asked to select whether they would like to receive one question per page, to select the weight and height unit of measure, to select which health assessment sections to complete, and whether the "User" prefers to focus on a specific category of medical conditions. The "User" will then start navigating the "Health Assessment" sections. The "User" will have the ability to transition from one section to the other. They can also skip questions and come back to them later. The system will display a message at the end of each section to transition the "User" from one section to the other. During the navigation, the "User" can see their progress in terms of the percentage of completion. If the "User" does not complete the "Health Assessment", the system will send weekly emails reminding the "User" to complete the "Health Assessment".

- Develop Release Plan: In this process, the product team develops a *Release Plan*, which is basically a phased deployment timeline that can be shared with the project's stakeholders. The length of each sprint is usually decided in this process. Some Scrum practitioners develop a *Product Roadmap* that is more strategic and high-level and a *Release Plan* that is more tactical and detailed.

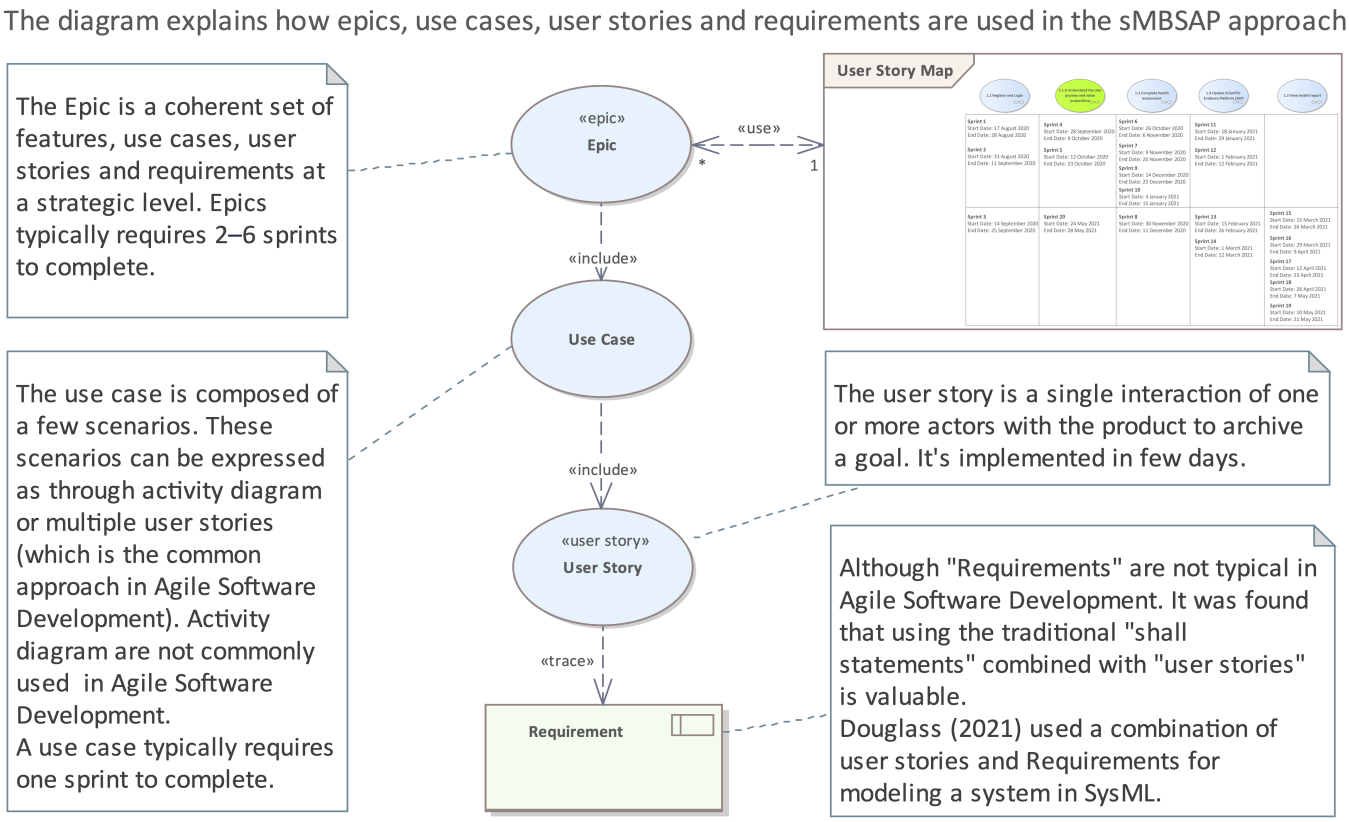

Figure 5. Relationship among Epics, Use Cases, *User Stories*, and *Requirements* in the sMBSAP method.

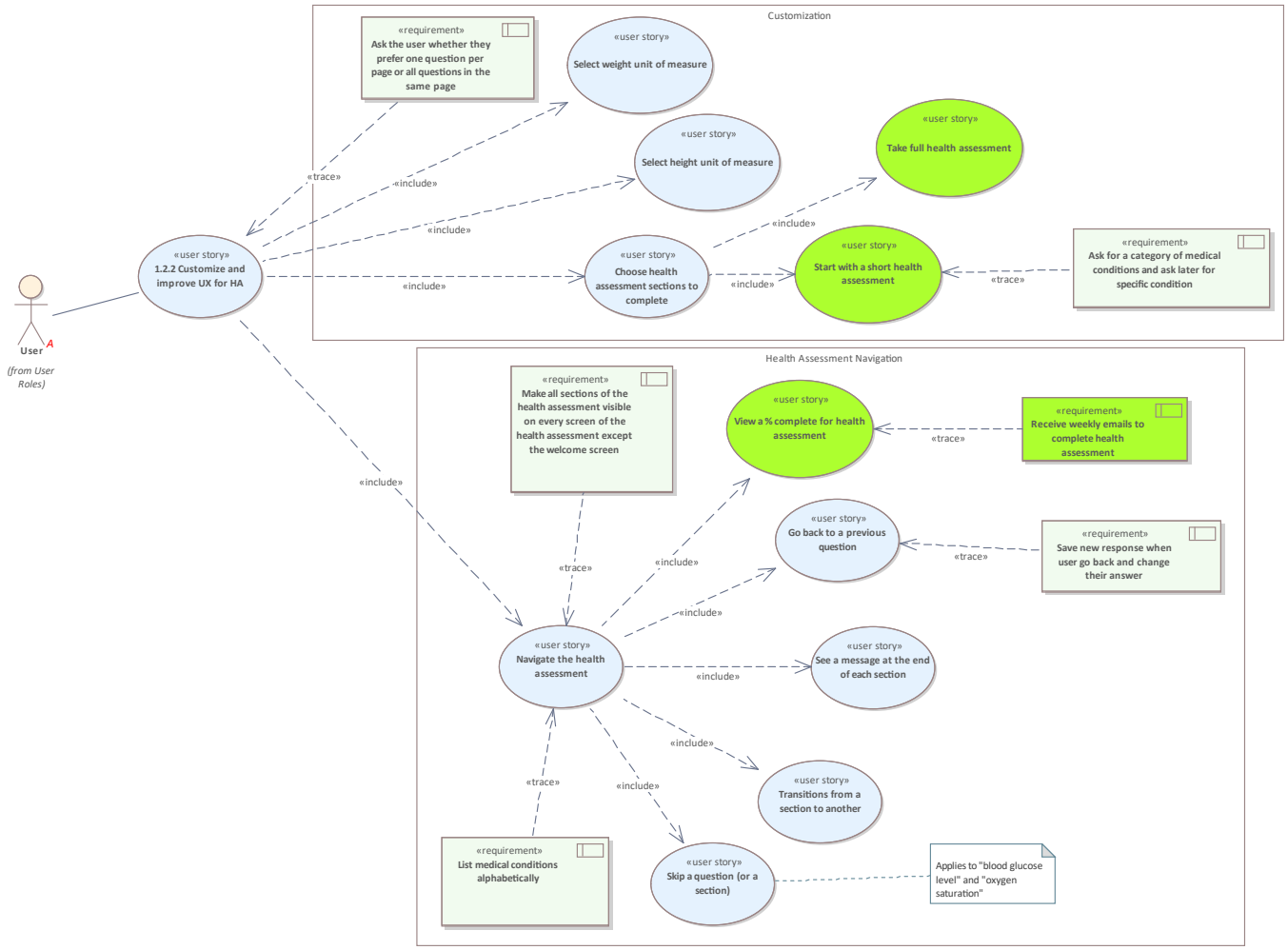

Figure 6. An example of Use Cases stereotyped as User Stories and *Requirements*.

3.4. Plan and Architect

This second phase consists of the processes related to planning, architecting, and estimating the *PBIs*. These processes are summarized below. It is important to note that although these processes are presented sequentially, in practice, they overlap, and the outcome of later processes serves as an input for former processes.

- Create and Update Backlog Items: In this process, *PBIs* (*User Stories*, *Requirements*, and tasks) and their related acceptance criteria are created or updated and incorporated into the *Product Backlog*. *User Stories* are designed to ensure that the project *Requirements* are clearly defined and can be entirely understood by all stakeholders. When a *User story* is committed, it can be broken down into specific tasks (or *Requirements* and tasks). Agile development tools can show the task list beneath the relevant *User Story*.
- Develop System Architecture: In this process, the *System Architect* progressively develops the system architecture. The system architecture is developed in lockstep with the *User Stories*. Both are used by the *Product Development Team* to execute the development work. The three main viewpoints progressively developed during this process are:
  - Operational Viewpoint (OV): The first progression is concerned with translating the *Project Scope Statement*, *Product Scope*, and *PBIs* (in any form that they are expressed) into an architectural model known as an *Operational Viewpoint (OV)*. This mapping is achieved with *Use Cases* and other object-oriented constructs. Several researchers, such as Lattanze [55], stressed the importance of starting with

a high-level view of the architecture before progressing to a more detailed design. The high-level view of the architecture is the primary purpose of the OV. The OV also defines the system's boundary and context. It also creates top-level partitioning (*Domains*), primary behaviors (*Use Cases*), and primary data content. With the aid of the model, the *System Architect* maps *User Stories* to *Domains* and *Use Cases*. The data model developed in this first progression is called "Conceptual Data Model (CDM)". This is the most abstract type of data model. Platform-specific information, such as data types, sequences, procedures, and triggers, are not included in the CDM. Because of its simplicity, it is useful for communicating ideas among different stakeholders. Data models can be developed with a number of notations, such as Information Engineering, IDEF1X, UML data modeling, and Entity Relationship notation.

The conceptual UML-based data model developed for the health tech system is shown in Figure 7. At this stage, the *System Architect* wants to capture and communicate the types of data (or "*Entities*") that the health tech system needs with the *Product Development Team*. These entities include the "User", "Health Assessment", "Report Subsections Decisions", "Medical Reference", and others. In addition to *Entities*, the CDM also captures the *Relationships*, i.e., how an *Entity* connects to other *Entities*. In the case of the health tech system, the "User" takes the "Health Assessment". Based on the results of the "Health Assessment", the "Report Subsections Decisions" will be displayed to the "User" and form the basis of the "Health Report". The "Report Subsections Decisions" rely on the "Medical Reference" for communicating the recommendations to the "User". The "Grocery Recommendations" are derived from "Report Subsections Decisions" and depend on both the "Nutrition and Ingredients" and "Medical Reference". As noted on the CDM, both "Nutrition and Ingredients" and "Medical Reference" are not exposed to the "User".

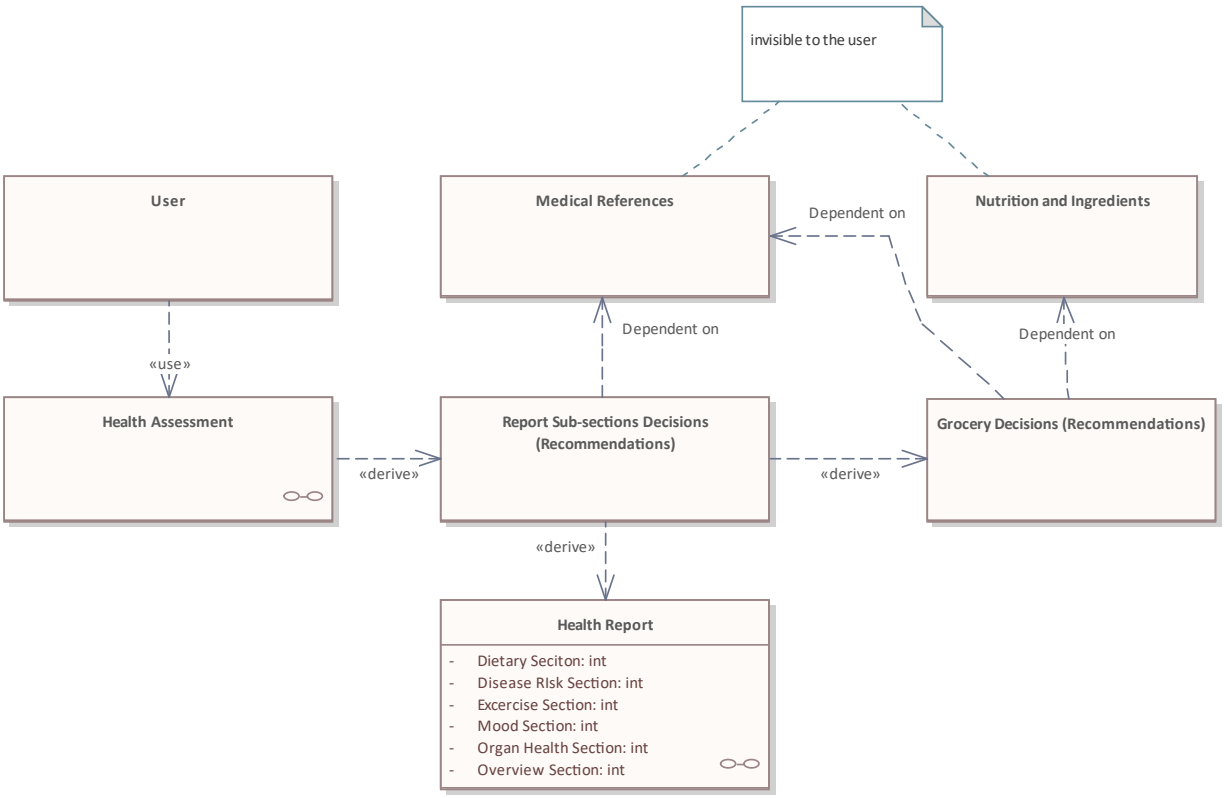

Figure 7. Conceptual Data Model for a health technology system.

– Functional/Logical Viewpoint (LV): The next progression transforms the OV into the *Logical/Functional Viewpoint (LV)*. This is where the design begins using UML class diagrams to define the details of system elements, functions, and data. The LV is a progressive elaboration on the perspectives of the OV and is molded mainly by decomposing *Domains* and *Use Cases* to develop structural and behavioral diagrams. The functional service specifications are developed and allocated to logical components and interfaces. The architectural layering is defined. The LV represents a functional definition of the technology- and product-agnostic system. The data model developed in this architecture iteration is called a "Logical Data Model (LDM)". The LDM defines the detailed *Structure* of a system's data elements and the relationships between them. It elaborates on the CDM introduced during the OV progression, but without going to the level of specifying the Database Management System (DBMS) that will be used. LDM forms the basis of the "Physical Data Model (PDM)". This model is commonly developed by using the UML Data Modeling notation. The logical UML-based data model developed for the health tech system is shown in Figure 8. As shown in Figure 8, the data elements "Medical References" and "Nutrition and Ingredients" contain UML attributes; the names and generic data types remain platform-independent. Platform-specific data types and other metadata that relate to a specific DBMS implementation are defined by the PDM.

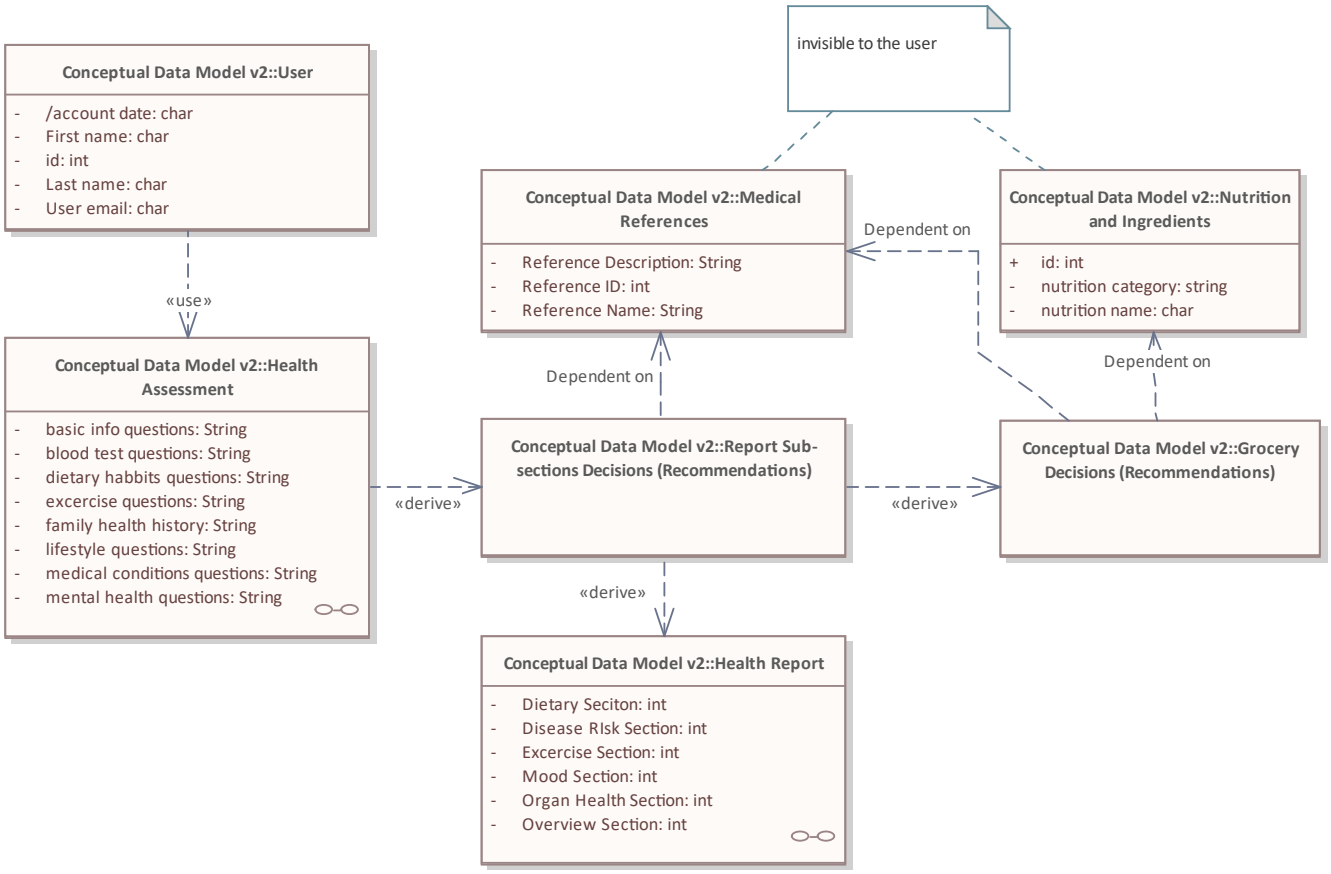

Figure 8. Logical Data Model for a health technology system.

– Physical Viewpoint (PV): The architecture modeling is completed by progressing from the LV to the *Physical Viewpoint (PV)*. The PV is the basis for the actual implementation of the full system or an increment of the system. To clarify the relationship between the LV and the PV, the former defines what is to be built, and the latter defines how it will be realized [13]. Accordingly, this architecture

iteration focuses on products and standards whose selection is paramount to reaching a physical design. The data model developed during the PV is called a "Physical Data Model (PDM)". A PDM graphically represents the *Structure* of data as implemented by a relational database schema. The ability to automatically generate the database schema from a PDM is a significant advantage of PDMs, in addition to presenting a visual abstraction of the database structure. This is made feasible by the amount of metadata that a PDM captures and its close alignment with aspects of the database schema, such as database tables, columns, primary keys, and foreign keys. The UML-based PDM developed for the health tech system is shown in Figure 9. Each table is represented by a UML Class; table columns, primary keys, and foreign keys are modeled by using UML attributes and operations. The DBMS type used in the system is PostgreSQL.

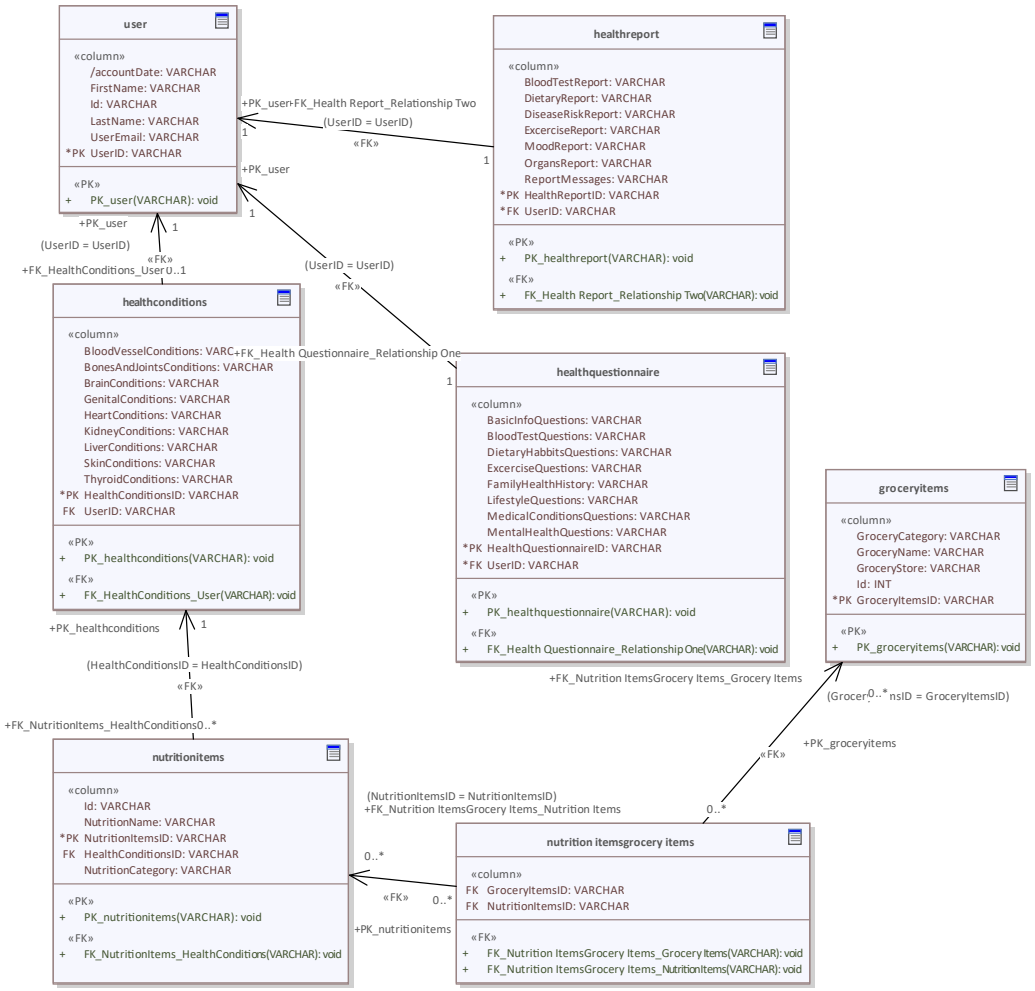

Figure 9. Physical Data Model for a health technology system.

It is important to note that each viewpoint is represented with several perspectives (within the viewpoints); the perspectives are largely derived from the fundamental elements of the architecture and the needs of the project. The proposed perspectives for the sMBSAP are the *Structure*, *Behavior*, *Data*, and *Requirements*, as shown in Figure 10. The importance of an adequate model *Structure* in achieving the full benefits of MBSE should be emphasized [13]. One way to group the content of each viewpoint into a set of perspectives that create a logical and easily searchable content framework is shown in Figure 11.

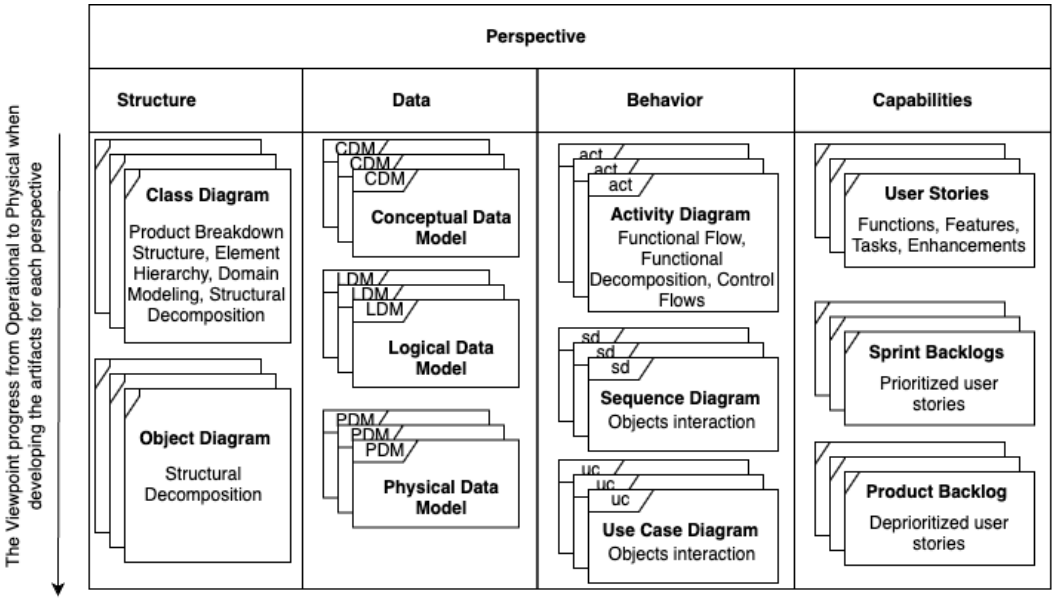

Figure 10. Organizational overview of an information model for the UML-based sMBSAP.

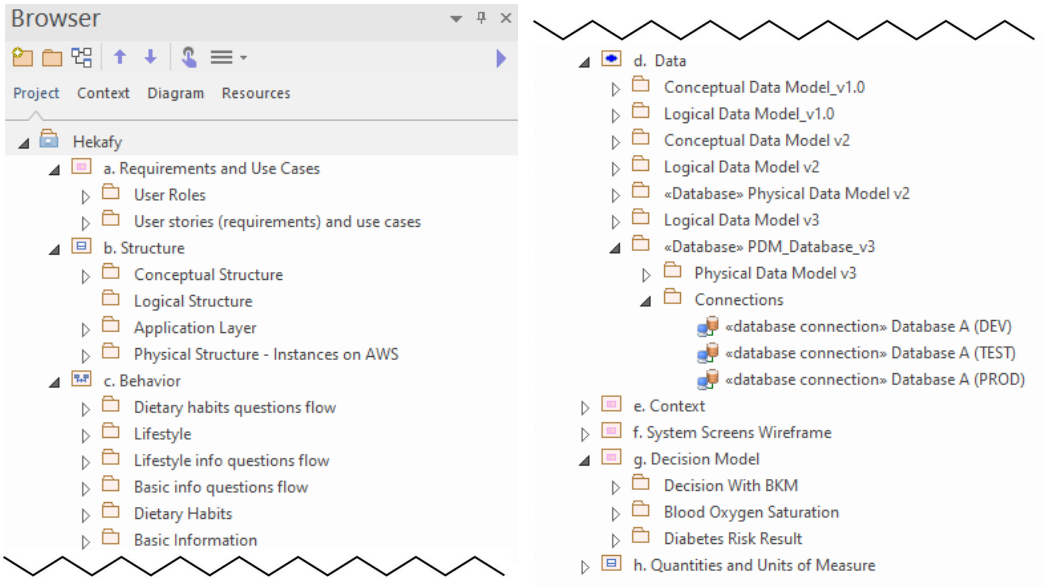

Figure 11. An illustration of organizing the sMBSAP artifacts in an MBSE tool.

- Commit User Stories: In this process, the project team commits to delivering the approved *User Stories* for a sprint. The committed *User Stories* are added to the *Sprint Backlog*. During the *Sprint/Architecture Planning*, the Scrum team may add further details to the *PBIs*.
- Estimate Backlog Items: In this process, the project team, supported by the *System Architect*, estimates the *PBIs* and estimates the effort required to develop the functionality described in each *PBI*.

3.5. Implement

This third phase is related to the execution of the activities required to develop the capabilities described by the *PBIs* to create the product. These processes are summarized below.

- Create Deliverables/Product Increments: In this process, the project team works on the items in the *Sprint Backlog* to create sprint deliverables. The project team's progress, measured in completed story points, is captured in an Agile development

tool. Planned versus actual story points are captured in the tool, in addition to marking an item as "done" when it is completed. The collected data are plotted on burnup charts and velocity fluctuation charts to allow the *Scrum Master* to monitor the project's health and make course corrections when needed.
It is important to note that creating deliverables/*Product Increments* may include activities such as project management, software engineering, continuous integration and testing, system configuration management, security, and other aspects. The sMBSAP is similar to the MBSAP in that the design modeled in the PV is built up in a prototype and goes through continuous integration and testing to assess its suitability against the required capabilities that are being addressed.

- Communicate Progress: In this process, the project team members update each other on their individual progress and any barriers that they may be facing. These updates occur through a short daily 15 min meeting, referred to as a *Standup Meeting*. The *System Architect* participates in these meetings and addresses any issues that the *Product Development Team* faces in relation to the system architecture.
- Groom Product Backlog: In this process, the prioritized *PBIs* are continuously updated and refined. A backlog grooming meeting is conducted to discuss any changes or updates to the backlog.
- Update System Architecture: In this process, the system architecture models are continuously updated and refined based on the progress and feedback from the project team. The results of the architecture changes or updates are discussed during the *Sprint/Architecture Review*.

3.6. Review and Retrospect

This fourth phase is concerned with reviewing the deliverables and work completed and identifying areas for improvement for future consideration. The processes of this phase are summarized below:

- Demonstrate and Validate Deliverables: In this process, the *System Architect* presents the updated system architecture to the project stakeholders. The project team then demonstrates the sprint deliverables that match the models to the stakeholders. These presentations and demos occur in a *Sprint/Architecture Review* meeting. This meeting aims to gain the acceptance of the delivered *PBIs* from the *Product Owner*.
  Screenshots from the health tech system product demo are shown in Figure 12. The product demo shows the four key steps in the "User" journey at a high level. In step 1, the "User" completes the registration process by entering their first name, last name, and email address, creating a password, and confirming it. After the "User" confirms their email address, they are redirected to the login page. In step 2, the "User" will be introduced to the "Health Assessment" and start completing its various sections. At the end of the "Health Assessment", the "User" will proceed to the third step, which is reviewing their "Health Report". Finally, in the fourth step, the "User" will receive the grocery items recommended for their health profile.
- Retrospect Sprint: In this process, the project team meets in a *Sprint Retro* meeting to discuss the lessons learned from the previous sprint. This information is recorded and should be used in future sprints. As a result of this meeting, some actions to improve performance or to make course corrections may be decided upon.

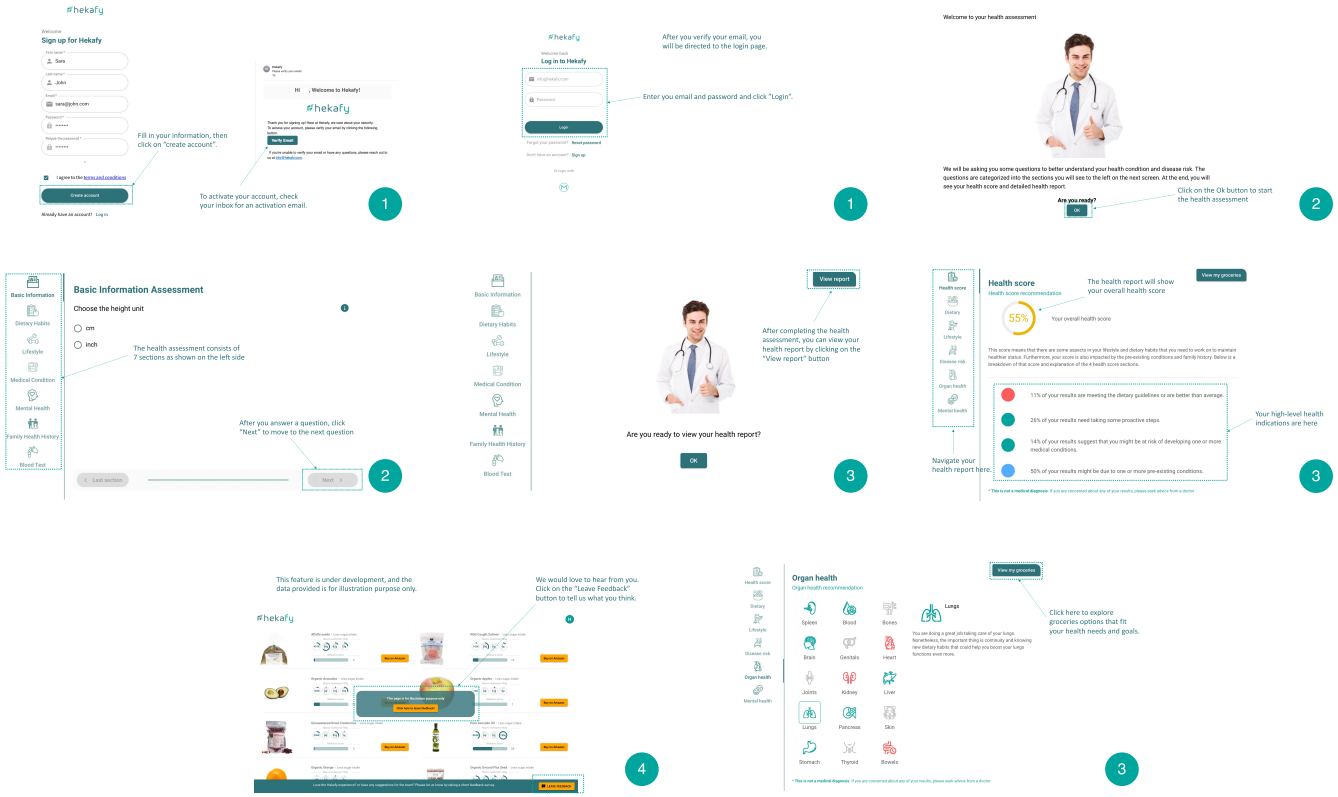

Figure 12. Screenshots from the health tech product demo.

3.7. Release

This fifth and final phase is about delivering the finally accepted deliverables to the customer. In addition, the lessons learned from the project are identified and documented. These processes are summarized below.

- Ship Deliverables and Architecture Models: In this process, approved and accepted deliverables are handed over to the concerned stakeholders. A formal transition document should be drafted and signed by the concerned stakeholders denoting the successful completion of the agreed-upon shippable part of the product. The architecture models are also handed over to the concerned stakeholders. The combined artifacts developed for the health tech system are shown in Figure 13.
- Retrospect Project: This is the final step in the project, in which the project team and stakeholders meet to identify and document the lessons learned for future implementation. This meeting is called the *Project Retrospective* meeting (or *Retro*).

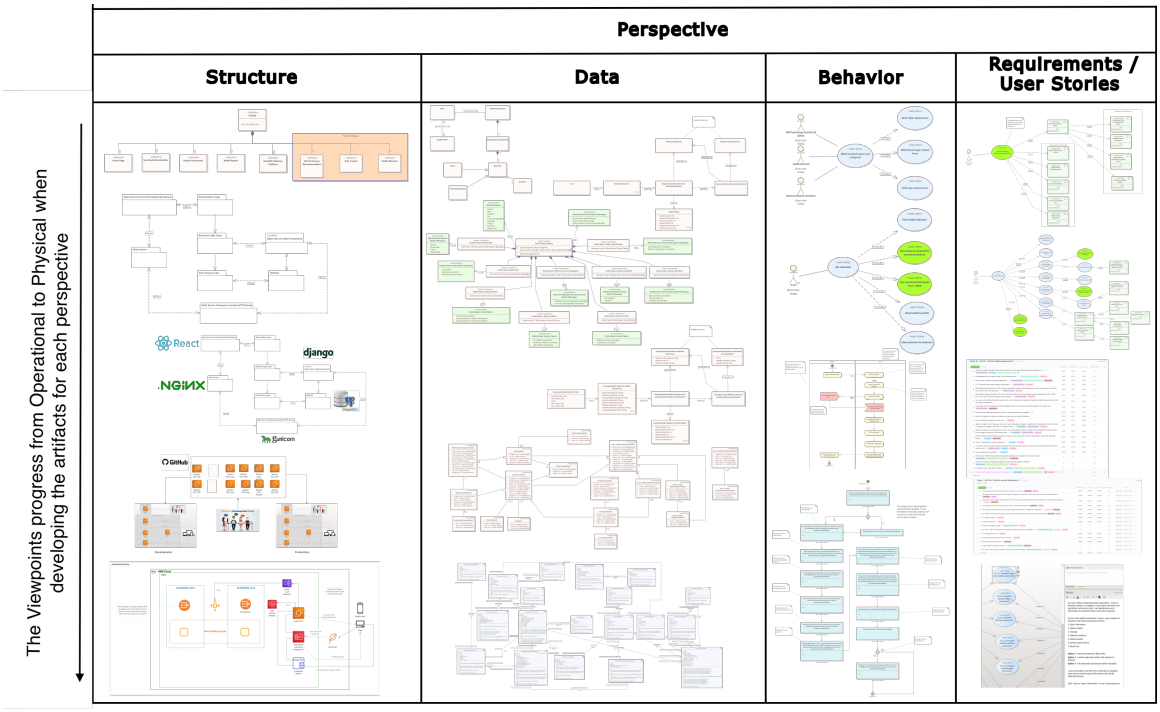

Figure 13. Combined sMBSAP artifacts for a health tech system.

4. Discussion

In addition to the main characteristics of Scrum and the MBSAP, the sMBSAP is also concerned with the engagement of the *Product Development Team* in customizing the MBSE tool, using UML-based and non-UML-based models to describe the system, and leveraging the built-in models (provided in some tools) to get to an initial version of the model more quickly. Moreover, the Scrum ceremonies are mapped and integrated into the entire sMBSAP lifecycle with some modifications. The sMBSAP has the same cyclic shape as Scrum to demonstrate that the development process is iterative. The iterative nature applies to both the product delivery and the system model construction.

Figure 14 shows a comparison among Scrum, MBSAP, and sMBSAP. The comparison reveals that Scrum and MBSAP have similarities, including a focus on collaboration, iterative and incremental development, and continuous improvement. In addition, both approaches value customer-centricity, prioritizing delivering a working product to the customer, and responding to change. These similarities have been passed down to the sMBSAP. However, there are also differences among Scrum, MBSAP, and sMBSAP. Scrum prioritizes working software as the primary measure of progress over system documentation. Scrum documentation does not follow a formal modeling language. While the MBSAP values a working product, it places a great emphasis on using models to capture and communicate system information. The sMBSAP, on the other hand, is a middle point, as it uses both formal and informal modeling languages to keep system information within the model. The model can be customized to keep the details of system architecture at a high level or comprehensive. Additionally, Scrum is primarily focused on software development. However, the MBSAP is more focused on systems engineering and the creation of high-level system models. The sMBSAP, on the other hand, is application agnostic and can be applied to software, defense, or other industries. As for project size, Scrum is primarily used for small to medium-sized projects. However, the MBSAP is more geared towards medium to large-sized projects. Finally, the sMBSAP can be customized to fit small to large projects.

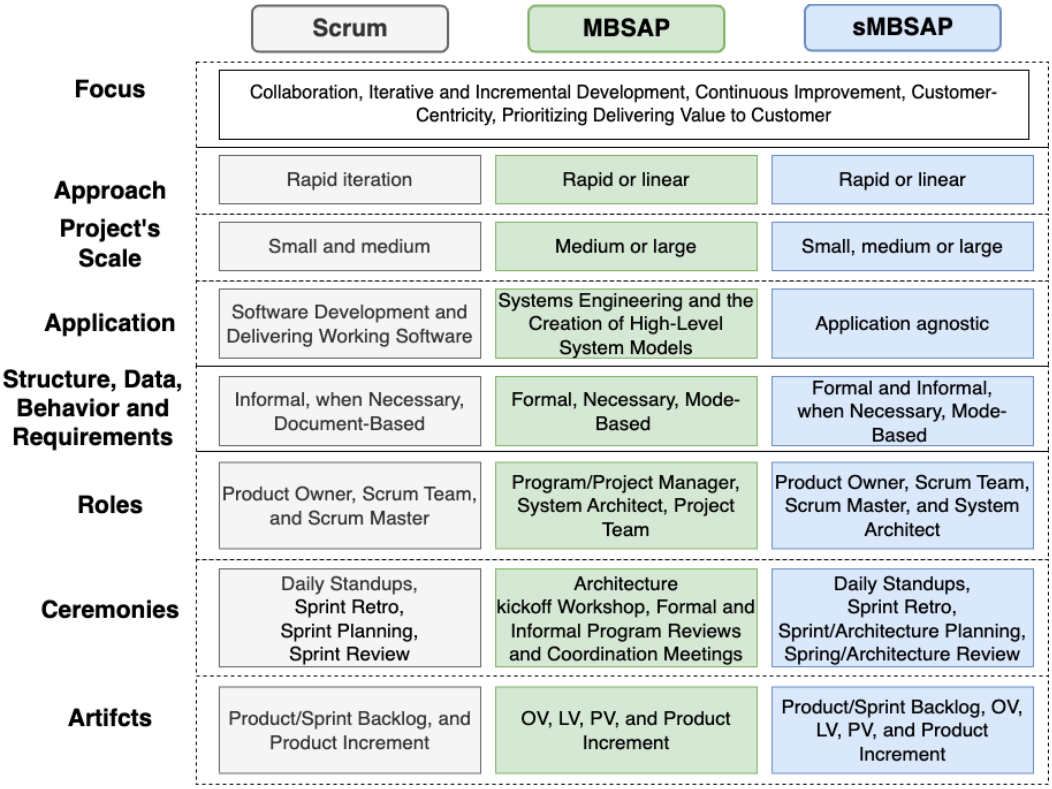

Figure 14. Comparison of Scrum, MBSAP, and sMBSAP.

Unlike traditional document-based methods, an MBSE tool is the key to handling, processing, and executing the data and information generated or collected during the system development process. Therefore, it is important to select the appropriate tool to create a modeling environment that fits the different kinds of data and information being processed. A proper MBSE tool can simplify the working process and accelerate the working efficiency. The MBSE tool used for architecting the health tech system in this pilot study was Sparx EA [54]. Sparx EA was selected due to its compatibility and readiness for software development models.

It is important to note, however, that in an MBSE-driven environment, having the best tools in the wrong environment would not contribute to project success. What contributes to project success is having the right group of individuals in product development. Even more crucial is how these people interact with one another. The other factor contributing to success is building a feedback loop with the customer to ensure that successful *Product Increments* are delivered. This feedback loop will open the door to embracing change, which always happens. These factors are inherited from both Scrum and MBSAP for sMBSAP, and they align well with the four values of the Agile Manifesto [47].

When a change is requested by the customer in the middle of a sprint, it is suggested to add the created *User Story* to the *Product Backlog* and reprioritize the *PBIs* rather than adding the *User Story* to the current *Sprint Backlog*. The benefit of this way of handling change is that it would avoid assuming that the development team would finish their work in progress and would be able to begin and finish the added *User Story* by the end of the sprint. The more assumptions a project has, the more risk exposure it has. Moreover, adding *User Stories* to an ongoing sprint would impact the monitoring of Estimation Reliability and Velocity.

During the execution of the phases of the sMBSAP approach, data were generated or collected from the beginning to the end of the health tech system development effort. Keeping the data and artifacts in one model made accessing and retrieving data easier compared to the process when using document-based methods. Tracking back the *Requirements* (*User*

Stories) or even performing simple simulation tasks for validation and verification was also beneficial. The key characteristics and benefits of implementing the sMBSAP include the following:

- The *System Architect* works closely, not in a silo, with the *Product Development Team* to (1) co-customize the MBSE tool at the beginning of the project to align with the needs of the project and the *Product Development Team*. The customization exercise is used as an MBSE infusion opportunity. In an MBSE-driven project, the *Product Development Team* is the first customer of the system architecture, and the Maintenance and Operations team is the end user, as they leverage the system architecture in operating software applications, monitoring system performance, making defect repairs, etc. The *System Architect* would need to work closely with different business and technical stakeholders from the customer organization to ensure that the model perspective and viewpoints are customized to fit their needs and the organization's standards.
  (2) It is also necessary to engage and educate the *Product Development Team* about the basic concepts and processes of the architecture and (3) to empower and support the *Product Development Team Members* (owners of core components of the system) to define discipline-specific MBSE methods. For example, a frontend developer develops a wireframe for the "Health report summary" including the screen elements, such as the following: buttons—"View my groceries"; dashboard indicators and messages—"Health score" and "Summary health report"; navigation sections—"Dietary", "Lifestyle", "Disease risk", "Organ health", and "Mental health"; finally, a scroll bar. The *System Architect* works closely with the frontend developer to ensure that every screen element is aligned with and mapped to *Requirements* or *User Stories*, as shown in Figure 15. The *System Architect* adds the relevant *Requirements* and *User Stories* to the wireframe. Throughout the process, the wireframe is refined and updated to reflect the intended use of each screen. The outcome of this collaborative effort is a model-based wireframe (an artifact unique to the sMBSAP).
- Selecting an MBSE tool with (1) built-in methodologies that support the transformation of current systems engineering practices into model-centric engineering practices and (2) built-in models that support specific *Use Cases* would help create a faster first iteration of the model. This fast turnaround increases the engagement of the *Product Development Team* and contributes to changing the perception that architecture modeling slows down the development process. For example, some MBSE tools have built-in Gantt charts, which can automatically display the schedule for sprints, and built-in and customizable dashboards that can be used to show the progress of a sprint. Moreover, the *Product Owner*, *Scrum Master*, and *Team Member* roles can all be supported, needless to say, by the role of the *System Architect*. Selecting and customizing the right MBSE tool will provide a cohesive collaboration and*Requirement* management platform.
- The sMBSAP enables the *System Architect* to use Agile terminologies that the *Product Development Team* understands. Implementing Agile concepts such as sprints, *Product Backlog*, *Epics*, and *User Stories* conveys a sense of familiarity to the *Product Development Team*, even if these concepts are implemented within the context of an MBSE and architecture-driven environment.
- The sMBSAP leverages the MBSE tool to combine the UML-based formal description of the system with non-formal models that fit the needs of the *Product Development Team*. Combining formal and non-formal modeling aids in addressing the usability challenge, as it gives more freedom to the *Product Development Team*. The value of this combination is to instill in the *Product Development Team* the concept of keeping all artifacts in one model. For example, wireframes are valuable visuals that are widely used in Agile software development. Integrating the non-UML-based wireframes in the sMBSAP approach could increase the engagement of the *Product Development Team* and the adoption of the sMBSAP.

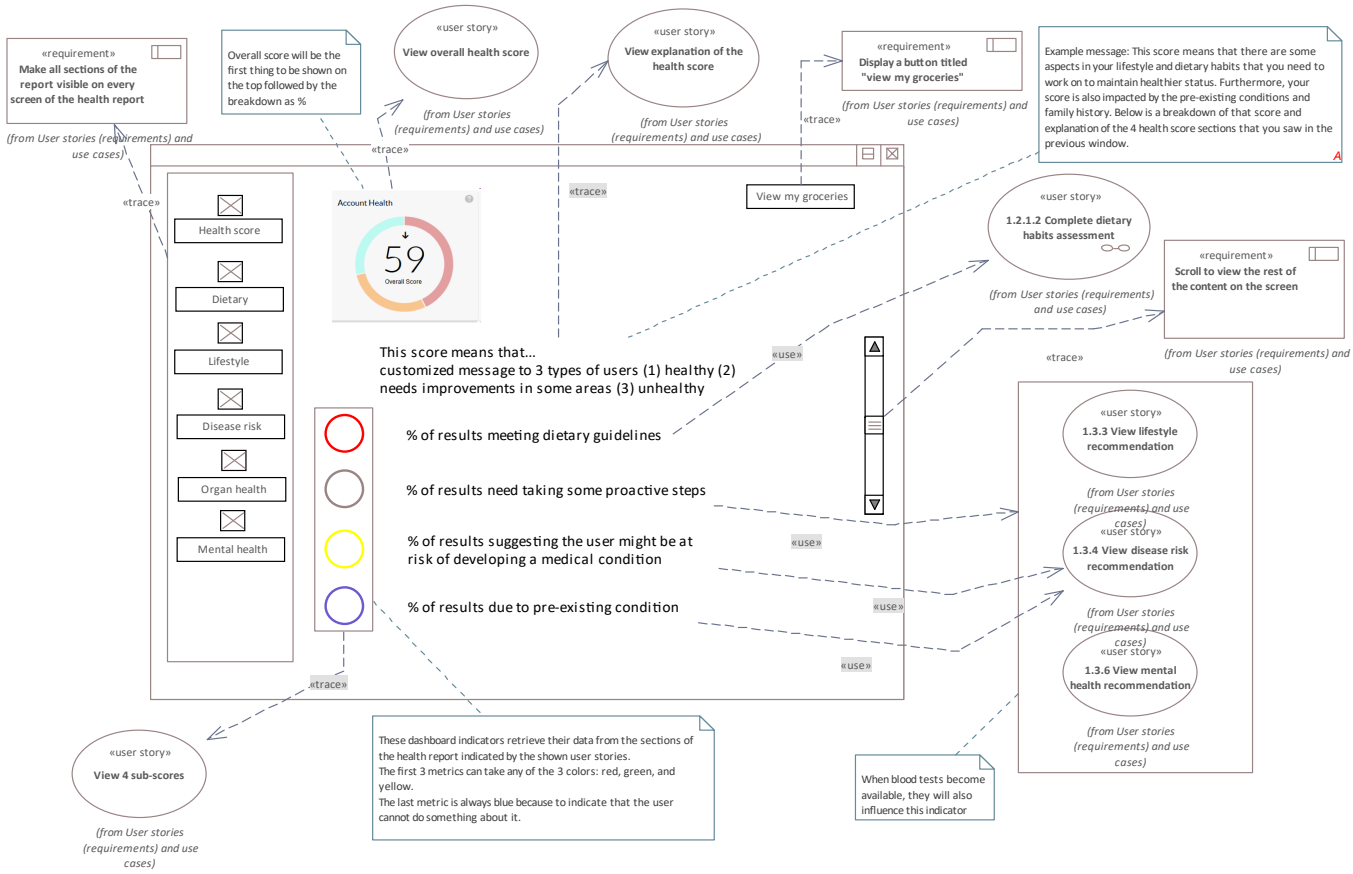

Figure 15. *Requirements* and user stories traced to elements of a wireframe diagram embedded in the sMBSAP model.

- The architecture effort progresses one sprint at a time. After the *System Architect* prepares the high-level end-to-end OV of the system, they focus only on the *Requirements* and the *Structural*, *Behavioral*, and *Data* perspectives of one sprint at a time. In that sprint, the OV is further elaborated into an LV, which will be progressively elaborated in future sprints. This approach could potentially be a step toward addressing the disconnect between how the system is conceptualized and how it is described, since the description is progressively elaborated over several weeks or months.
- The *Requirements* and *User Stories* will be traced to the various perspectives of the system model throughout the subsequent iterations or viewpoints. In Use Case diagrams, *Requirements* are traced to *User Stories* that are modeled using a stereotyped Use Case. *Requirements* can then be shown on other models to trace the implementation of *Requirements* and *User Stories* (a *Requirement* or *User Story* is created once and used multiple times across the model). This approach simplifies the *traceability* process and bypasses the need to develop and maintain a document-based *Requirement Traceability Matrix*. Model-based *Requirement Traceability* could be a meeting ground between those who devalue *Traceability* and those who want to align with procurement/reporting practices.
- The iterative benefits of Agile combined with an integrated model of artifacts, as proposed in the sMBSAP, could be a practical happy medium between light documentation enthusiasts and those who value heavy documentation. The centralized management of artifacts makes the sMBSAP approach suitable for projects that value a working product and, at the same time, are keen to have more manageable, accessible, and retrievable documentation via a system architecture model.

On the other hand, there are a few challenges related to the adoption of the sMBSAP. Some of these challenges include the perception that MBSE is performed by a tool, although

several researchers explained that MBSE is more than just a tool [38,43,44]. Selecting an MBSE tool without a deep understanding of user needs is another challenge that may impede the adoption of the sMBSAP. Transitioning to a model-based software engineering approach requires a high level of executive support, which may not be always present. Finally and most importantly, adopting the sMBSAP requires a considerable investment in training because of its steep learning curve. Organizations may implement organizational change management initiatives to facilitate organizational adoption, but such organization-wide initiatives themselves require investment and management support.

5. Conclusions and Future Work

In this paper, an integration of the Agile and MBSE approaches has been proposed. The new approach, termed the Scrum Model-Based System Architecture Process (sMBSAP), uses the same cyclic approach of Scrum and the MBSAP.

The sMBSAP approach includes five main artifacts: *Product/Sprint Backlog*, *Operational Viewpoint (OV)*, *Logical/Functional Viewpoint (LV)*, *Physical Viewpoint (PV)*, and *Product Increment*, as well as four roles: *Product Owner*, *Scrum Master*, *System Architect*, and *Product Development Team*. The five sMBSAP phases include Initiate, Plan and Architect, Implement, Review and Retrospect, and Release.

Both Scrum and the MBSAP focus on collaboration and continuous improvement. Both approaches also value customer-centricity and prioritize delivering a working product to the customer. These similarities have been passed down to the sMBSAP. The sMBSAP customizes the artifacts to keep system information within the model. The sMBSAP is application agnostic and can be applied to software or other industries. As for project size, the sMBSAP can be customized to fit small to large projects. The sMBSAP approach was validated through a pilot study to develop a health technology system over one year.

The preliminary results have shown that the proposed approach contributed to achieving the desired system development outcomes and, at the same time, generated complete system architecture artifacts that would not have been developed if Agile alone had been used. The highlights of the sMBSAP approach and benefits observed during the implementation can be summarized as follows: (1) The *System Architect* works closely, not in a silo, with the *Product Development Team* to customize, empower, and educate the team to get the best out of the architecture model; (2) selecting an MBSE tool with built-in methodologies and models helps create a faster first iteration of the model; (3) the sMBSAP enables the *System Architect* to use Agile terminologies that the *Product Development Team* understands; (4) the MBSE tool enables the *System Architect* to combine formal and informal modeling to gradually shift the mindset of the Agile team towards MBSE; (5) the architecture effort progresses one sprint at a time; (6) the *Requirements* and *User Stories* will be traced to the various perspectives of the system model throughout the following iterations or viewpoints; (7) the sMBSAP is a practical middle ground between light documentation enthusiasts and those who value heavy documentation.

The promising results observed while using the model are a step towards closing the gap between Agile and MBSE. The sMBSAP offered a practical and operational method for achieving the desired and potentially better outcomes compared to either approach alone. In parallel, this research shows that the sMBSAP is more aligned with federal and state regulations, which promote Agile in its systems engineering guidelines while requiring a proper set of system documentation.

The continuation of this research project includes quantitatively comparing the impact on the system development objectives when using the sMBSAP compared to Scrum. Specifically, the subsequent step includes conducting a quasi-experimental study to compare Scrum and the sMBSAP in terms of system development performance metrics, as measured by the estimation reliability, productivity, and defect rate.

Author Contributions: Conceptualization, M.H., J.M.B. and D.R.H.; methodology, M.H.; software, M.H.; validation, D.R.H.; formal analysis, M.H.; investigation M.H.; data curation, M.H.; writing—original draft preparation, M.H.; writing—review and editing, D.R.H.; visualization, M.H. and D.R.H.; supervision, D.R.H. and J.M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Sharing research data may be restricted due to intellectual property on the part of the health tech company whose system development was used to conduct this research.

Acknowledgments: The research described in this publication would not have been possible without leveraging the system development effort at Hekafy. The lead author would like to acknowledge the support of the co-founders of Hekafy: Reem Olaby, Mohamed Farid, and Aneesh Panoli.

Conflicts of Interest: The lead author is one of the co-founders of Hekafy Inc.

Abbreviations

The following abbreviations are used in this manuscript:

ASD	Adaptive Software Development
AUP	Agile Unified Process
CDM	Conceptual Data Model
DBSE	Document-Based Systems Engineering
DoDAF	Department of Defense Architecture Framework
DSDM	Dynamic Systems Development Method
FDD	Feature-Driven Development
FEAF	Federal Enterprise Architecture Framework
LDM	Logical/Functional Data Model
LV	Logical Viewpoint
MAGIC	Munich Agile MBSE Concept
MBSAP	Model-Based System Architecture Process
MDSD	Model-Driven Systems Development
MBSE	Model-based Software Engineering
NIST	National Institute of Standards and Technology
OPM	Object-Process Methodology (Dori)
OOSEM	Object-Oriented Systems Engineering Method (INCOSE)
OV	Operational Viewpoint
PBI	Product Backlog Items
PDM	Physical Data Model
PV	Physical Viewpoint
RAD	Rapid Application Development
RUP	Rational Unified Process
RUP SE	Rational Unified Process for Systems Engineering (IBM)
SAFe	Scaled Agile Framework
SDLC	Software Development Life Cycle
SA	State Analysis (JPL)
SE	Systems Engineering
sMBSAP	Scrum Model-Based System Architecture Process
TOGAF	The Open Group Architecture Framework
UML	Unified Modeling Language
XP	Extreme Programming

References

1. Hooshmand, Y.; Adamenko, D.; Kunnen, S.; Köhler, P. An approach for holistic model-based engineering of industrial plants. In Proceedings of the International Conference on Engineering Design, Vancouver, BC, Canada, 21–25 August 2017; Volume 3, pp. 101–110.
2. Friedenthal, S.; Moore, A.; Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*, 3rd ed.; Morgan Kaufmann: Burlington, MA, USA, 2015. [CrossRef]

3. Basha, N.M.J.; Moiz, S.A.; Rizwanullah, M. Model based software development: Issues & challenges. *Int. J. Comput. Sci. Inform.* **2013**, *3*, 84–88. [CrossRef]
4. Call, D.R.; Herber, D.R. Applicability of the diffusion of innovation theory to accelerate model-based systems engineering adoption. *Syst. Eng.* **2022**, *25*, 574–583. [CrossRef]
5. Cao, L.; Ramesh, B. Agile software development: Ad hoc practices or sound principles? *IT Prof.* **2007**, *9*, 41–47. [CrossRef]
6. Altahtooh, U.A.; Emsley, M.W. Is a challenged project one of the final outcomes for an IT project? In Proceedings of the Hawaii International Conference on System Sciences, Waikoloa, HI, USA, 6–9 January 2014; pp. 4296–4304. [CrossRef]
7. Muganda Ochara, N.; Kandiri, J.; Johnson, R. Influence processes of implementation effectiveness in challenged information technology projects in Africa. *Inf. Technol. People* **2014**, *27*, 318–340. [CrossRef]
8. Yeo, K.T. Critical failure factors in information system projects. *Int. J. Proj. Manag.* **2002**, *20*, 241–246. [CrossRef]
9. Turner, R. Toward Agile systems engineering processes. *Crosstalk J. Def. Softw. Eng.* **2007**, *20*, 11–15.
10. Salehi, V.; Wang, S. Munich Agile MBSE Concept (MAGIC). In Proceedings of the Design Society: International Conference on Engineering Design, Delft, The Netherlands, 5–8 August 2019; Volume 1, pp. 3701–3710. [CrossRef]
11. Riesener, M.; Doelle, C.; Perau, S.; Lossie, P.; Schuh, G. Methodology for iterative system modeling in Agile product development. *Procedia CIRP* **2021**, *100*, 439–444. [CrossRef]
12. Bott, M.; Mesmer, B. An analysis of theories supporting Agile scrum and the use of scrum in systems engineering. *Eng. Manag. J.* **2020**, *32*, 76–85. [CrossRef]
13. Borky, J.M.; Bradley, T.H. *Effective Model-Based Systems Engineering*; Springer: Berlin/Heidelberg, Germany, 2019. [CrossRef]
14. Douglass, B.P. *Agile Model-Based Systems Engineering Cookbook*; Packt: Birmingham, UK, 2021.
15. Bouillon, E.; Güldali, B.; Herrmann, A.; Keuler, T.; Moldt, D.; Riebisch, M. Leichtgewichtige Traceability im agilen Entwicklungsprozess am Beispiel von Scrum. *Softwaretechnik-Trends* **2013**, *33*, 29–30. [CrossRef]
16. Lethbridge, T.; Singer, J.; Forward, A. How software engineers use documentation: The state of the practice. *IEEE Softw.* **2003**, *20*, 35–39. [CrossRef]
17. Voigt, S.; Hüttemann, D.; Gohr, A.; Große, M. Agile Documentation Tool Concept. In *Developments and Advances in Intelligent Systems and Applications;* Springer: Berlin/Heidelberg, Germany, 2018; pp. 67–79.
18. Ruparelia, N.B. Software development lifecycle models. *ACM SIGSOFT Softw. Eng. Notes* **2010**, *35*, 8–13. [CrossRef]
19. Rather, M.A.; Bhatnagar, M.V. A comparative study of software development life cycle models. *Int. J. Appl. Innov. Eng. Manag.* **2015**, *4*, 23–29.
20. Tsai, B.Y.; Stobart, S.; Parrington, N.; Thompson, B. Iterative design and testing within the software development life cycle. *Softw. Qual. J.* **1997**, *6*, 295–310. [CrossRef]
21. Kossiakoff, A.; Biemer, S.M.; Seymour, S.J.; Flanigan, D.A. *Systems Engineering Principles and Practice*; John Wiley & Sons: Hoboken, NJ, USA, 2020.
22. Dora, S.K.; Dubey, P. Software development life cycle (SDLC) analytical comparison and survey on traditional and Agile methodology. *Natl. Mon. Ref. J. Res. Sci. Technol.* **2013**, *2*, 22–30.
23. Schmidt, C. *Agile Software Development Teams*; Progress in IS; Springer: Berlin/Heidelberg, Germany, 2015. [CrossRef]
24. Khong, L.; Yu Beng, L.; Yip, T.; Soofun, T. Software development life cycle AGILE vs traditional approaches. In Proceedings of the International Conference on Information and Network Technology, Chennai, India, 28–29 April 2012; Volume 37, pp. 162–167.
25. Vijayasarathy, L.R.; Turk, D. Agile software development: A survey of early adopters. *J. Inf. Technol. Manag.* **2008**, *XIX*, 1–8.
26. U.S. Government Accountability Office. *Agile Assessment Guide: Best Practices for Agile Adoption and Implementation*; Technical Report GAO-20-590G; U.S. Government Accountability Office: Washington, DC, USA, 2015
27. Anand, R.V.; Dinakaran, M. Issues in scrum Agile development principles and practices in software development. *Indian J. Sci. Technol.* **2015**, *8*, 1–5. [CrossRef]
28. DIGITAL.AI. 16th State of Agile Report. 2022. Available online: https://info.digital.ai/rs/981-LQX-968/images/AR-SA-2022-16th-Annual-State-Of-Agile-Report.pdf (accessed on 14 April 2022).
29. Schwaber, K. *Agile Project Management with Scrum*; Microsoft Press: Unterschleissheim, Germany, 2004.
30. Satpathy, T. *A Guide to the Scrum Body of Knowledge (SBOK™ Guide)*, 3rd ed.; SCRUMstudy: Avondale, AZ, USA, 2016.
31. Akif, R.; Majeed, H. Issues and challenges in scrum implementation. *Int. J. Sci. Eng. Res.* **2012**, *3*, 1–4.
32. Buffardi, K.; Robb, C.; Rahn, D. Learning agile with tech startup software engineering projects. In Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education, Bologna, Italy, 3–5 July 2017; pp. 28–33. [CrossRef]
33. Ghezzi, A.; Cavallo, A. Agile business model innovation in digital entrepreneurship: Lean startup approaches. *J. Bus. Res.* **2020**, *110*, 519–537. [CrossRef]
34. Kuchta, D. Combination of the earned value method and the Agile approach—A case study of a production system implementation. In *Intelligent Systems in Production Engineering and Maintenance*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 87–96. [CrossRef]
35. Atlassian. Jira | Issue & Project Tracking Software | Atlassian. 2019. Available online: https://www.atlassian.com/software/jira (accessed on 14 April 2022).
36. ClickUp™. Available online: https://www.clickup.com.html (accessed on 28 February 2023).
37. Brambilla, M.; Cabot, J.; Wimmer, M. *Model-Driven Software Engineering in Practice*; Springer: Cham, Switzerland, 2017. [CrossRef]

38. Bonnet, S.; Voirin, J.L.; Normand, V.; Exertier, D. Implementing the MBSE cultural change: Organization, coaching andlessons learned. In Proceedings of the INCOSE International Symposium, Seattle, WA, USA, 13–16 July 2015; Volume 25, pp. 508–523. [CrossRef]
39. Walden, D.D.; Roedler, G.J.; Forsberg, K. INCOSE systems engineering handbook version 4: Updating the reference for practitioners. In Proceedings of the INCOSE International Symposium, Seattle, WA, USA, 13–16 July 2015; Volume 25, pp. 678–686. [CrossRef]
40. Estefan, J.A. *Survey of Model-Based Systems Engineering (MBSE) Methodologies*; Technical report; INCOSE MBSE Initiative: San Diego, CA, USA, 2008.
41. Zimmerman, P. A review of model-based systems engineering practices and recommendations for future directions in the department of defense. In Proceedings of the Systems Engineering in the Washington Metropolitan Area Conference, Chantilly, VA, USA, 3 April 2014.
42. Wang, L.; Izygon, M.; Okon, S.; Wagner, H.; Garner, L. Effort to accelerate MBSE adoption and usage at JSC. In Proceedings of the AIAA SPACE, Long Beach, CA, USA, 13–16 September 2016. [CrossRef]
43. Young, K.G. Defense space application of MBSE-closing the culture chasms. In Proceedings of the AIAA SPACE Conference and Exposition, Pasadena, CA, USA, 31 August–2 September 2015. [CrossRef]
44. Noguchi, R.A. A roadmap for advancing the state of the practice of model based systems engineering for government acquisition. In Proceedings of the INCOSE International Symposium, Orlando, FL, USA, 20–25 July 2019; Volume 29, pp. 678–690. [CrossRef]
45. Kim, S.Y.; Wagner, D.; Jimenez, A. Challenges in applying model-based systems engineering: Human-centered design perspective. In Proceedings of the INCOSE Human-Systems Integration Conference, Biarritz, France, 11–13 September 2019.
46. Hadar, I.; Sherman, S.; Hadar, E.; Harrison, J.J. Less is more: Architecture documentation for agile development. In Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering, San Francisco, CA, USA, 25 May 2013; pp. 121–124. [CrossRef]
47. Manifesto for Agile Software Development. Available online: http://agilemanifesto.org (accessed on 27 March 2023).
48. Stettina, C.J.; Heijstek, W. Necessary and neglected? An empirical study of internal documentation in agile software development teams. In Proceedings of the ACM International Conference on Design of Communication, Pisa, Italy, 3–5 October 2011; pp. 159–166. [CrossRef]
49. Pasuksmit, J.; Thongtanunam, P.; Karunasekera, S. Towards just-enough documentation for agile effort estimation: What information should be documented? In Proceedings of the IEEE International Conference on Software Maintenance and Evolution, Luxembourg, 27 September–1 October 2021; pp. 114–125. [CrossRef]
50. Prause, C.R.; Durdik, Z. Architectural design and documentation: Waste in agile development? In Proceedings of the International Conference on Software and System Process, Zurich, Switzerland, 2–3 June 2012; pp. 130–134. [CrossRef]
51. Rubin, E.; Rubin, H. Supporting agile software development through active documentation. *Requir. Eng.* **2011**, *16*, 117–132. [CrossRef]
52. Selic, B. Agile documentation, anyone? *IEEE Softw.* **2009**, *26*, 11–12. [CrossRef]
53. Slack. Available online: https://slack.com (accessed on 28 February 2023).
54. Sparx Systems. Enterprise Architect 15.2 User Guide. Available online: https://sparxsystems.com/enterprise_architect_user_guide/15.2/ (accessed on 28 February 2023).
55. Lattanze, A.J. *Architecting Software Intensive Systems: A Practitioners Guide*; Auerbach Publications: Boca Raton, FL, USA, 2008.