applied sciences

Article

Compiling Requirements from Models for Early Phase Scope Estimation in Agile Software Development Projects

Lina Bisikirskienė, Lina Čeponienė *, Mantas Jurgelaitis, Linas Ablonskis and Eglė Grigonytė

Department of Information Systems, Faculty of Informatics, Kaunas University of Technology, 44249 Kaunas, Lithuania; lina.bisikirskiene@ktu.lt (L.B.); mantas.jurgelaitis@ktu.lt (M.J.); linas.ablonskis@ktu.lt (L.A.)

* Correspondence: lina.ceponiene@ktu.lt

Abstract: Inadequate early scope estimation is a common problem in software projects, leading to failures in meeting project requirements. Agile projects usually do not concentrate on a comprehensive requirements analysis and specification before the start of the project, making scope assessment difficult. This paper presents the methodology for facilitating a more accurate early estimation of project scope, based on requirements information gathered in various forms (requirements models and textual descriptions) during the requirements workshop. The requirements from different sources are compiled into one list and reconciled, since they are prepared by a number of participants in the requirements workshop using different notations (UML diagrams, SysML models, Story map) and may have differences in the vocabulary. Reconciliation encompasses the unification of vocabulary, as well as the identification and the removal of overlaps in requirements. The final list of requirements is used to estimate the scope of the project in story points. The estimate can be presented to the client and used as a basis for the project contract. A case study on the application of the proposed methodology is presented, using the animal shelter information system as a development project. It demonstrates that the methodology is viable and can facilitate the gathering of a more extensive set of requirements, thus ensuring a more detailed scope estimation.

Keywords: project scope estimation; Agile; requirements workshop; UML; SysML

check for updates

Citation: Bisikirskienė, L.; Čeponienė, L.; Jurgelaitis, M.; Ablonskis, L.; Grigonytė, E. Compiling Requirements from Models for Early Phase Scope Estimation in Agile Software Development Projects. *Appl. Sci.* **2023**, *13*, 12353. https://doi.org/10.3390/app132212353

Academic Editors: Frederico Branco, José Martins and Henrique Mamede

Received: 30 October 2023
Revised: 8 November 2023
Accepted: 11 November 2023
Published: 15 November 2023

1. Introduction

Requirements engineering in Agile software development deals with various challenges, including poorly specified requirements, which result in inaccurate scope estimation [1]. This is especially prevalent in early phase project scope estimation, which must be performed before the software development project starts. During this early phase, initial requirements must be obtained from the client. Based on them, the scope of the product must be estimated and agreed upon to proceed further with the project execution. Estimation must be performed from a limited amount of data, since stakeholders are willing to allocate a limited amount of time and personnel to preliminary elicitation of the product requirements. However, the scope estimation must be as precise as possible, because it is used to develop a contract with the client and serves as a basis for more detailed requirement elicitation if stakeholders commit to the project.

In Agile software development projects, capturing domain knowledge can be challenging [2]. During requirement elicitation, stakeholder communication plays a crucial role. Although in Agile projects, requirements are commonly specified using User stories and Story maps, additional visualizations can be helpful in facilitating discussions, in understanding and communicating the requirements to the client and within the development team [2–4]. These visualizations in the form of various diagrams can significantly enrich the definition of the system under development and enhance the understanding of the project scope. If we choose not only to document user stories that reflect individual requirements, but to model domain knowledge including its behavioral aspects, we acquire a broader view of future product behavior and structure [2]. UML [5] diagrams are one of the popular choices for modeling software requirements. SysML [6] is also a popular notation for systems of systems modeling.

Although Agile software development methodologies do not explicitly suggest the use of models to specify requirements, there are various approaches that discuss the benefits of introducing modeling into Agile development [4,7,8]. When introducing the usage of models in Agile requirement elicitation, it must be considered that these models can be prepared as sketches. They cover the details from different points of view, as they are prepared by different members of the team. Requirements can be overlapping, not aligned in the terms of the domain vocabulary, and even contradicting. Therefore, the alignment and reconciliation of these models must also be considered, ensuring that the information depicted in the models is useful in the project and can also serve as a basis for project scope estimation. Support for diverse sources of requirements and identifying overlapping requirements is crucial for resource and stakeholder management [9], risk identification and mitigation [10,11], and ensuring that the final product meets its intended objectives. It contributes to better stakeholder involvement and higher-quality results, thus improving stakeholder satisfaction [12,13].

The goal of this paper is to facilitate an early-phase Agile software development project scope estimation based on the elicitation and reconciliation of requirements presented in various forms. The objectives are to propose a methodology for early phase scope estimation, to develop a tool for facilitating the application of the methodology, and to apply it in a case study of a real-life software development project. We aim to demonstrate that the proposed methodology enables gathering a more detailed and reconciled requirements list, which then can be used for scope estimation in the early phase of Agile project. In order to facilitate common understanding of project scope and complexity, the proposed methodology encompasses gathering requirements in the form of UML and SysML models, Story maps and Requirement lists during the requirements workshop, compiling and reconciling requirements list from all provided sources, and estimating project scope by assessing each requirement in story points.

There exists a variety of scope estimation (also known as effort estimation) approaches that can be divided into two main groups [14,15]: algorithmic (or data-based) and non-algorithmic (or expert-based). Algorithmic approaches usually rely on large amounts of historic data and mathematical solutions, and currently also rely on the application of artificial intelligence and machine learning. On the other hand, the non-algorithmic approaches are based on expert experience and deduction. In Agile projects, expert-based methods, such as planning poker and story points, are commonly used [15,16]. They support basic Agile principles, including the encouragement of teamwork, team consensus, and team-specific estimations. In this paper, we propose an expert-based methodology for software scope estimation, which is constructed on early requirements gathered during the requirements workshop [16,17]. To properly engage stakeholders of various competencies in the requirements workshop, we encourage using various visual models for recording requirements and support the extraction of requirements into one common set from various sources: Story maps, requirements lists, UML [5] Use case models, UML Activity models, UML Class models, and SysML [6] Requirement models. Due to requirements coming from various potentially disjoint media, we perform requirement reconciliation to derive a requirements list, which is then used for story points-based scope estimation.

The tool implementing the principles of the proposed methodology was developed and employed in the presented case study of the animal shelter information system scope estimation. During the scope estimation demonstrated in the case study, the requirements were gathered not only from Story map, but also from various other models. This enabled the acquisition of a more detailed requirement list and a more detailed scope evaluation.

The rest of the paper is structured as follows. The second section analyzes related work in the areas of Agile requirements elicitation, modeling usage, and scope estimation. The third section presents the proposed early-phase scope estimation methodology and a detailed description of its main steps: the requirements workshop, compilation of requirements from various sources, and scope estimation. The fourth section describes the tool implementation of the proposed methodology, which provides functionality for importing requirements, facilitating their reconciliation and scope estimation. A case study representing the application of the proposed methodology in the early phase scope estimate of the pet shelter information system is analyzed in the fifth section. The last section concludes the results of the case study and discusses future work.

2. Related Work

This paper focuses on two main areas of interest: requirements engineering [16,18–22] and scope (or effort) estimation [23–27]. In the context of Agile project scope estimation, two important aspects are examined in more detail, early-phase requirements elicitation and scope estimation based on these early requirements, as well as the benefits of the introduction of the modeling paradigm in both [19,21,22].

While the Agile methodology offers a flexible development approach with a focus on individuals, collaboration, working software, and rapid responses to changes [12], the process of requirements elicitation begins with the project concept clarification [28]. This step involves identifying project stakeholders [29] and establishing goals, and also using various techniques, such as interviews, focus groups, brainstorming, and workshops, to gather initial requirements [12,28]. During requirements elicitation, the aspects of risks identification [10,11], stakeholder communication, and human resources management [9,29] should also be considered. Proposals for Agile requirements elicitation exist, outlining the need for tailored approaches [1,16,18,19], but only a few of them mention early-phase requirements elicitation. Agile may not always accommodate different representations, though textual notation may not always be an appropriate tool for describing different artifacts, like data models, user flows, or the user interface [2,12]. Just as in classic requirements engineering, in Agile, challenges concerning miscommunication might still occur. To address the challenges of representing artifacts that are not easily conveyed through user stories, solutions exist proposing to incorporate modeling with Agile practices [19,21,22]. Strategies for the elicitation and management of requirement [2,12,18] must be considered in the context of early scope estimation, as it relies on a backlog or a requirements list, which needs to be developed rapidly and employed to accurately measure scope. Therefore, we analyze in more detail the approaches that introduce requirement elicitation and management strategies [18,19,22], and the modeling for structuring the requirement elicitation processes in Agile projects [19,22].

A framework for improved software requirements elicitation in SCRUM [18] covers three elicitation phases. In the pre-elicitation phase, it utilizes a joint requirements document for identifying, breaking down, and prioritizing requirements based on stakeholder feedback. During mid-elicitation, an initial scope document and priority list of a high-level requirements are used to further decompose and refine requirements using mind mapping techniques. This process involves breaking down each requirement into as many branches as possible for both functional and non-functional aspects, ultimately resulting in the creation of user stories. The post-elicitation phase is performed during development for finalizing and validating user stories derived from mind maps. These user stories are included into the product backlog, where the stories are corrected and refined by the product owner and the development team for inclusion in the sprint backlog. This framework outlines that the hybrid of different approaches and the involvement of multiple stakeholders improves Agile software requirements.

Another method for the elicitation, documentation, and validation of requirements (MEDoV) [19] is tailored for Agile and Lean software development. The method supports decision-making processes, and during its first phase, business objectives and KPIs are identified. It employs UML Use case, Class, and Activity diagrams for requirement modeling. The developed "to be" and "as is" models can also be used for communication and validation purposes, thus increasing the clarity of requirements with minimal impact on agility.

An MBSE approach to Agile [22] utilizes both the SCRUM and the model-based system architecture to leverage the benefits of both methods. This combination enables the development of an architecture model, which includes structural, data, behavior, and capability perspectives. The approach facilitates the iterative model development and allows one to shift the perspective of an Agile team towards MBSE, enabling the tracing of requirements and user stories between various perspectives and iterations of the model. It also supports the development of documentation that is especially relevant to government agencies.

The introduction of a more structured approach, be it through modeling or employing various techniques for capturing, refining, and reconciling requirements involving different stakeholders and business analysts, allows knowledge gaps to be captured, and requirements to be validated, reconciled, and aligned [12,13]. The different perspectives provided by different people, and in some cases the use of different notations, allow us to develop a clearer picture and a more comprehensive backlog [2,12,13,18]. Once a comprehensive backlog is formed, albeit one that is not entirely detailed, most high-level requirements should be present and can be used in the scope estimation process.

In Agile, not only requirement elicitation, but also scope and effort estimation present a unique set of challenges, especially in dealing with evolving incomplete requirements [15,23]. As the topics of effort, scope, cost, and size estimation are closely related, and the terms scope estimation and effort estimation are often used interchangeably, we analyzed research on any of these topics in the area of Agile software development, especially its early phase. Precise estimation is challenging due to the requirements' variability, as the backlog may not be complete. Research on the introduction of machine learning techniques in scope estimation is underway; e.g., ref. [26] introduces the use of the GPT-2 pre-trained model for estimating the effort (in Story Points) required to implement the product backlog, but is intended primarily to be used during sprint planning. The analysis on improving expert-based methods [27] is also relevant, which determined that factors such as developer knowledge, experience, and the complexity and the impact of changes on the system directly affect the precision of effort estimation. The incorporation of models can be also used in estimating the effort and scope. The use of models is more prevalent in development practices employing a more robust documentation, such as COSMIC [20,21], Use Case Points [30,31], or Function Points [32]. COSMIC can be used to derive functional size measures from UML software artifacts, such as Use case model, Activity, Sequence, Class, Package, and Component diagrams [20]. In approaches combining Agile and COSMIC methods, there are also proposals to derive COSMIC metrics from user stories [21]. The Use Case Points-based method Know-UCP [30] utilizes the UML Use case model as a basis for deriving Use Case Points from UML diagrams to estimate the size of the software. Use Case Points can even be adapted to the development of IoT systems [31]. Function Points are another metric used to determine the size of software, and there exist approaches based on Function Points, such as [32], which heavily rely on modeling. The analyzed approaches apply these estimation techniques during the development process to obtain more accurate measurements of the remaining effort and scope of the project. Additionally, proposals exist that use models in estimation and are tailored for specific stages of software development, e.g., testing the effort of object-oriented systems can be determined using UML Class diagrams in [33]. As testing efforts can be determined early, early scope estimation is extremely beneficial to determine scope before committing to the project. In the project bidding, which is an early phase of the project, the estimation of the scope is a crucial step [34,35]. It serves as a foundation for determining whether product or project development is feasible given limited resources and in communicating the complexity of the project both internally in the team and to the client. Unfortunately, most scope estimation methods focus on estimation when planning sprints or releases, and are not tailored to estimation in the early phase of the project [15,35]. We analyzed specific early scope estimation methods [23–25] more thoroughly, as these methods are specialized for dealing with an incomplete backlog and high-level requirements.

In the research on early phase cost estimation for Agile projects in the US DoD [23], a requirements-based complexity metric (based on [36]) is utilized. The authors conducted an examination of 20 Agile projects and concluded that a cost estimation based entirely on initial requirements is possible. High-level functional requirements and external interfaces are the main factors for estimating the project scope. The continuation of this research is also presented in [24], where the authors have provided an updated estimation method, this time focusing on effort and schedule estimation models for sizing Agile projects prior to the proposal. The proposed models were applied to 36 Agile projects, from a DoD backlog spanning 11 years. The authors determined that the initial software requirements that include high level functional and external interface requirements were directly related to the final Agile development effort and time, thus demonstrating that the effort could be estimated at earlier stages of development.

An effort and cost estimation approach that uses decision tree techniques and story points is presented in [25]. Estimation is tailored for Agile software development and is performed using decision tree, random forest, and Adaboost techniques or a combination of all three. These hybrid models can produce accurate estimates when trained on historical data. However, the authors determined that the completion time and cost estimate depend on effort, measured in story points, and team velocity.

While scope estimation is important to determine the remaining scope during software development, the use of early scope estimation techniques can serve as a crucial indicator of project feasibility and complexity. Unfortunately, the majority of the methods are not tailored to early project stages and require a heavier investment of time and expertise from developers [35]. Specific requirement elicitation techniques need to be applied, not only to rapidly develop backlog, but also to reconcile, validate, and align requirements with business goals [1,12]. On the other hand, machine learning and data-based methods can be used to support developers in scope estimation, but such methods rely heavily rely on historical data and will not take over team decision-making processes in Agile projects anytime soon [25], as Agile scope estimation is based on team velocity and requires the active involvement of team members in decision making. Expert-based methods, which usually rely on story points, are more relevant in Agile projects and are more commonly used [15,35]. Therefore, we proposed the scope estimation methodology, which elaborates both early-phase requirements elicitation techniques, which employ modeling, and application of expert-based estimation principles, when team members collaboratively reach the consensus on requirements assessments in story points.

3. The Early Phase Project Scope Estimation Methodology

The proposed methodology (Figure 1) is intended for the early stages of the project, in which an initial gathering of requirements is performed. The first step in the process is collaborative collecting of requirements during the requirements workshop, also known as the Lean Requirements Workshop [16,17]. Key stakeholders of the project must participate in the short (1–2 days) and intensive workshop, during which requirements are collected in various forms and then compiled into one set and reconciled. This reconciled set is used for scope estimation—each requirement is evaluated using story points and coefficients of clarity. The resulting scope estimate can be presented to the client and used as a basis for the contract. The gathered requirements are further analyzed in more detail and used for preparing the product backlog.

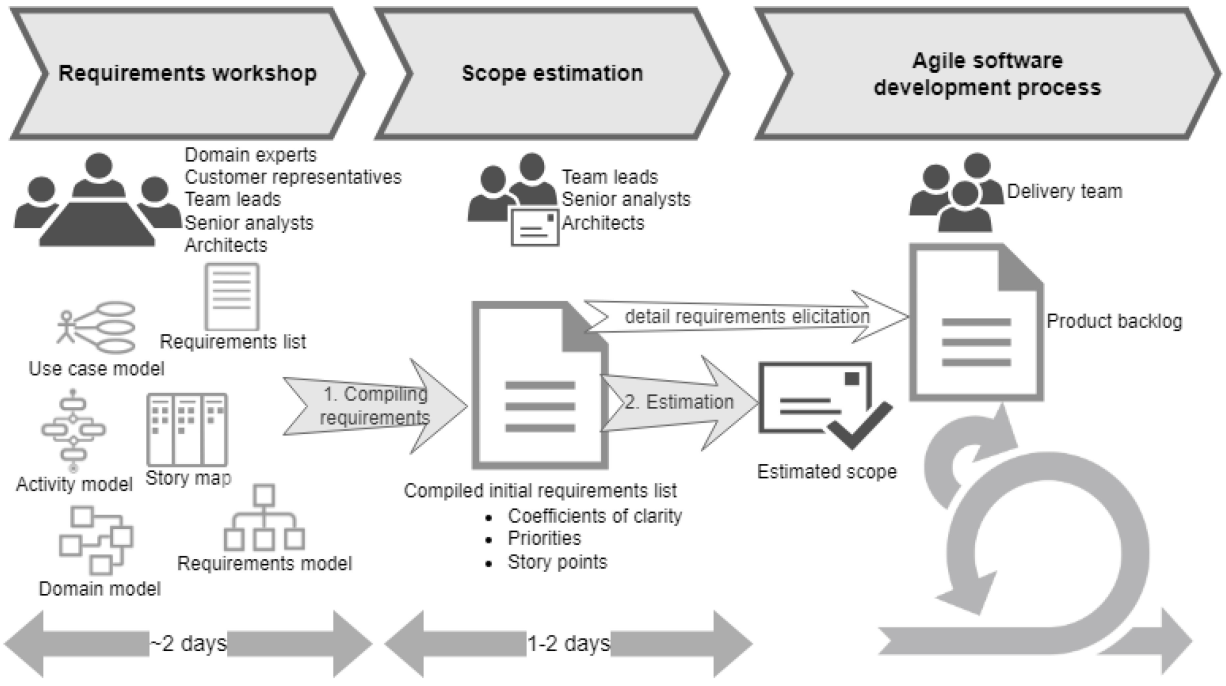


Figure 1. Overview of the early phase scope estimation methodology.

3.1. Requirements Workshop

The requirements workshop is a collaborative session conducted to gather, refine, and clarify requirements for a project or product development. It is a technique commonly used in Agile and Lean methodologies to streamline the requirements collection process. The workshop involves different stakeholders, such as business sponsors, domain experts, actual users, product owners, developers, designers, testers, business analysts, architects, etc. This diversity ensures that various perspectives are considered. Workshops are usually time-boxed, meaning that they have a limited duration, often ranging from a few hours to a day or two. This time constraint helps maintain focus and prevents the workshop from becoming overly lengthy and unproductive. The goal of the requirements workshop is to create a common understanding among stakeholders and team members about the project goals and scope. Participants engage in various interactive activities to discuss, document, and prioritize requirements. These activities might include brainstorming, user story mapping, process flowcharting, and collaborative discussions. Workshops often utilize visual aids such as whiteboards, sticky notes, and diagrams to help participants better understand and visualize requirements. Participants who are familiar with modeling languages can prepare various diagrams, using popular notations, such as UML or SysML. Visual representations can make complex ideas easier to understand and communicate. The variety of visual model representations ensures that each workshop participant can properly get involved and can contribute to preparing an initial list of requirements. The most used visualizations are the Story Map, UML Use case model, UML Activity model, and SysML Requirements model. If the participants have more technical experience, they may use the UML Class model. If the team does not use visualizations but only compiles lists of requirements, e.g., in Excel spreadsheets or Gannt Chart based tools, then the risk of wasting more time, missing process steps, users, risks, etc., increases. The assumption that modeling is time-consuming is incorrect if the level of detail of the models is properly defined. The recommendations for achieving the optimal level of detail for the models prepared during the requirements workshop are presented in Table 1.

Table 1. Requirements sources.

Source of Requirements	Analyzed Source Elements	Additional Recommendations for the Source
Story map	User, Epic, User story, Priority	Avoid design or technical constraints details.
UML Use case model	Actor, Use case, Association, Include, Extend, Package	Use Actors; Group Use cases with Packages; Avoid creating chains with more than one Include or Extend relationship.
UML Activity model	Swimlane (Activity Partition), Action, Activity, Object Node, Object Flow, Control Flow, Guard	Use Activities to decompose complex processes; Actions in the Activities should be of Use case level, not going into the detail of the Use case internal scenario; Avoid creating an Activity hierarchy with more than 3 levels; Use Actors for representing Swimlanes; Avoid specifying unnecessary exceptions, concentrate on happy path scenarios; Avoid different types of Actions, as differences do not bring value in high-level requirements.
UML domain (Class) model	Class, Association (also including composition and aggregation), Enumeration, Package	Identify a high-level picture (only domain entities and their Associations);Group elements using packages, if needed.
Requirements list	ID, Actor, Name, Description, Priority, Category	Define hierarchy using ID; Group requirements by Category; Use Actors.
SysML Requirement model	Requirement, Package	Group requirements using packages if needed; Form hierarchy, but not more than 2 levels, to avoid too much detail.

The main difference between the preparation of models in Agile and Waterfall projects is their detail. This is because the purpose of models in Agile projects is to facilitate a common understanding of the project and requirements guidelines, as the details will change over time. Waterfall projects require precision and detailed information already in the analysis phase, and this phase often lasts from one to several months. In Agile projects, the time to agree on the scope of the project and the main requirements for the first phase of the minimum viable product (MVP) is very limited.

In the proposed methodology, there is no limit to how many and what models team members can create during the requirements workshop. The goal is not to limit the participants, but to give them the opportunity to effectively synchronize the available information, fill in the gaps, and move to the scope assessment stage. This process is very fast, and the goal is not to obtain all the details, but to ensure wide coverage from various viewpoints and general mutual agreement. In Agile projects, the details are worked out as the project progresses.

3.2. Compiling Requirements List

When the workshop is finished, each of the participants in the project scope assessment has information that needs to be coordinated with the other participants. Information consists of lists of requirements, visual models, comments, and restrictions. Collecting and consolidating these requirements into the requirements specification is a complex process. This process is further complicated by the fact that people see and record different things while participating in the workshop. The consolidation process must ensure that their vision and understanding of the requirements are harmonized and allow a proper assessment of the scope of the project. Sometimes, for Scrum-based projects, only a Story map is created, but in most cases, it is not enough if one wants to have a clear picture of the scope. Therefore, other visual models should also be introduced.

The process of compiling requirements from various sources into requirements specification is presented in Figure 2. In our proposed process, when collecting requirements into the requirements specification, two options are possible: to analyze source models (Steps S1.1–S1.6) and gather requirements from them or to introduce requirements directly (Step S2). Since the process can be repeated as many times as needed, additional sources of requirements or manually added requirements can be introduced at any time. Five types of source models can be used to gather requirements: Story map, UML Use case model, UML Activity model, UML domain (Class) model, SysML Requirement model. Additionally, a simple Requirements list can also be used as a source. The order in which the source models are analyzed is not important, as the primary model is later selected to align the requirements.

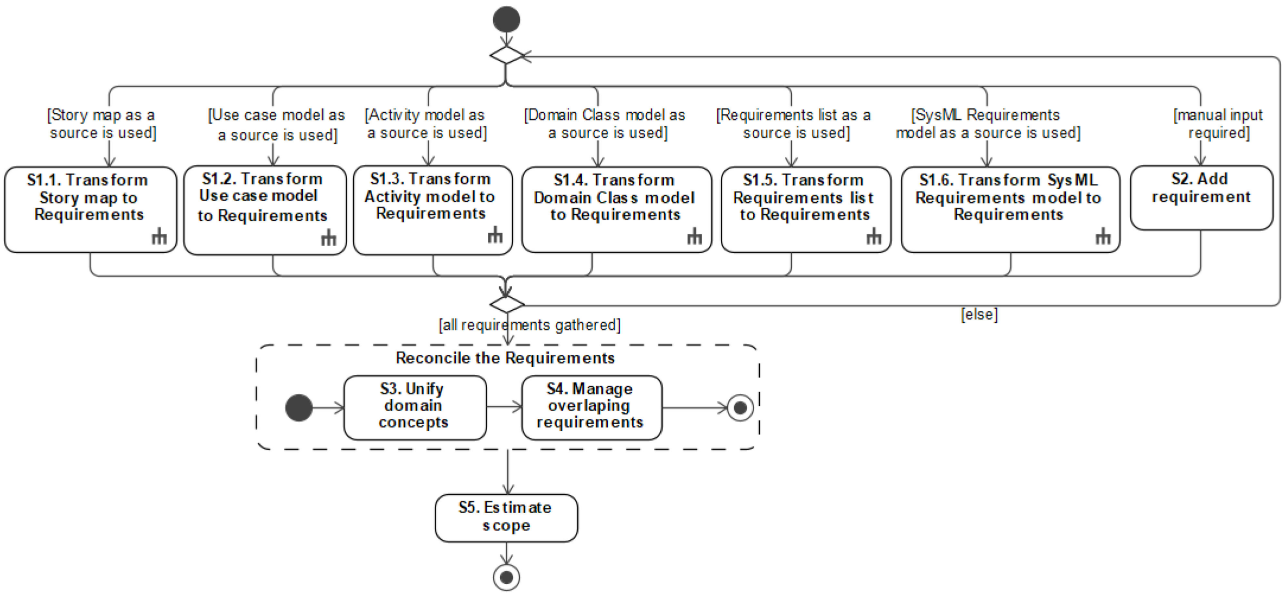

[Story map as a source is used]
[Use case model as a source is used]
[Activity model as a source is used]
[Domain Class model as a source is used]
[Requirements list as a source is used]
[SysML Requirements model as a source is used]
[manual input required]
S1.1. Transform Story map to Requirements
S1.2. Transform Use case model to Requirements
S1.3. Transform Activity model to Requirements
S1.4. Transform Domain Class model to Requirements
S1.5. Transform Requirements list to Requirements
S1.6. Transform SysML Requirements model to Requirements
S2. Add requirement
[all requirements gathered]
[else]
Reconcile the Requirements
S3. Unify domain concepts
S4. Manage overlaping requirements
S5. Estimate scope

Figure 2. The process of compiling requirements and estimating scope.

Once the source model is analyzed, the elements are identified, which can be used to form the requirements. Each model defines a certain set of elements which are chosen to ensure that the high-level information can be modeled and then transformed into requirements specifications. All source models can include comments or constraints, attached to any model element, then the requirement into which this model element is transformed will have a comment attribute, where constraint/comment from the model is recorded. In Table 1, the lists of elements for each model and additional recommendations are provided, which complement the basic principles of model development (not analyzed here) for ensuring the adequate use of models and their elements during the high-level requirements identification.

When the source model is analyzed, the requirements are transformed from the available elements using the proposed algorithms (steps S1.1–S1.6. of the process in Figure 2). This is the first step in which all available information about the project is aggregated into one set of requirements.

3.2.1. Compilation of Requirements from Various Sources

Compiling the requirements into a requirements specification is a step that allows the team to start a discussion about the scope of the project. The main problems encountered at this stage are the incompatibility of terms and the omission of functionality. Irrespective of the competences of the team members, the joint contribution makes it possible to manage the mentioned risks more effectively.

Each model used as a source of requirements has its own list of elements (defined in Table 1) that participate in the formation of a requirement. However, the principles of how the requirement is formulated are common to all models. The template by which the requirement text is formed is as follows:

[<Actor>] [can <Action> [if <Condition>]] [in <Context> context].

The text colors can be used for easier identification of the elements of requirement: Actor is written in blue, Action in green, and Context in red font. The main elements of the Requirement template, the additional Requirement attributes, and the sources of Requirements are presented in the metamodel in Figure 3.

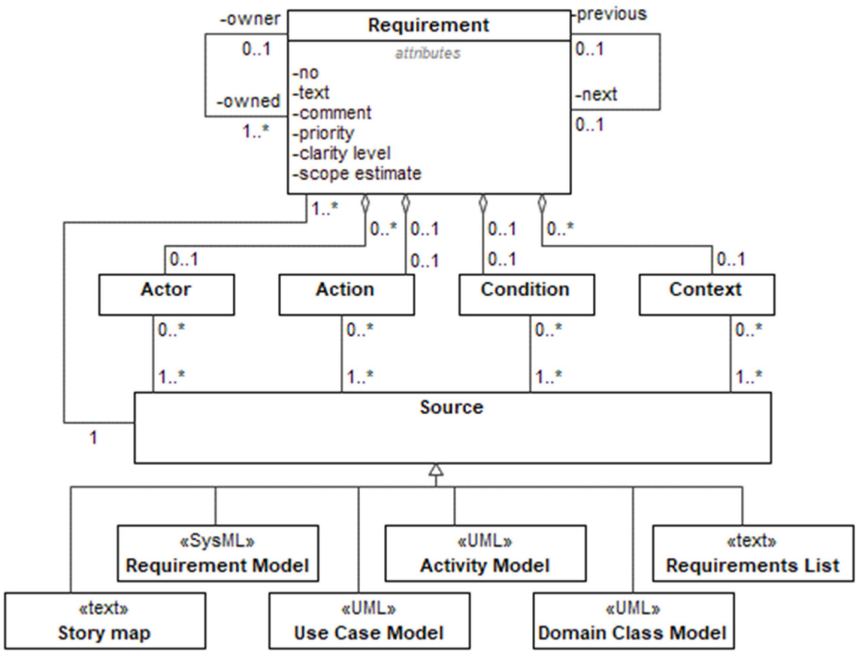


Figure 3. Requirements metamodel and related source models.

All elements of the Requirement template are optional, but at least one element of Actor, Action, or Context type must exist to form a Requirement. In the template, the <Action> element text is not always straightforwardly mapped from the source model, as it can be formed from several elements of the model. Additionally, the Requirement may have information about the priority, comment, owner (for identifying hierarchy), and previous and/or next elements (for identifying sequences). This information is extracted from source models where it is available. The algorithms for forming the Requirements from source models are presented in Figures 4–9.

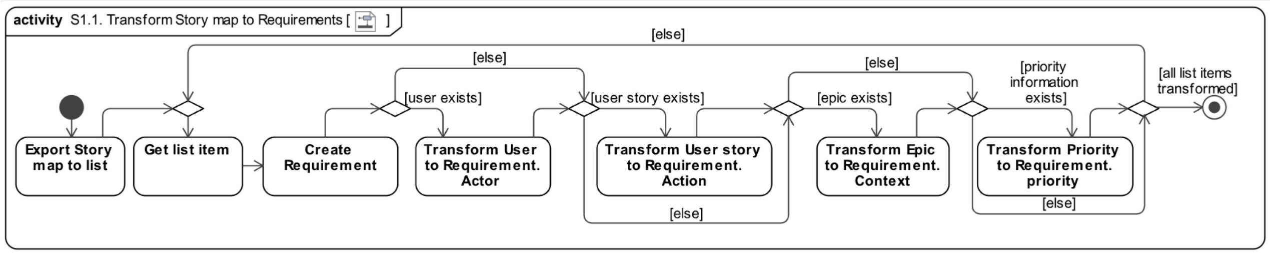

activity S1.1. Transform Story map to Requirements [ ]
Export Story map to list
Get list item
Create Requirement
Transform User to Requirement. Actor
Transform User story to Requirement. Action
Transform Epic to Requirement. Context
Transform Priority to Requirement. priority
[else]
[else]
[user exists]
[user story exists]
[else]
[else]
[epic exists]
[priority information exists]
[else]
[all list items transformed]

Figure 4. Algorithm for Story map transformation into Requirements.

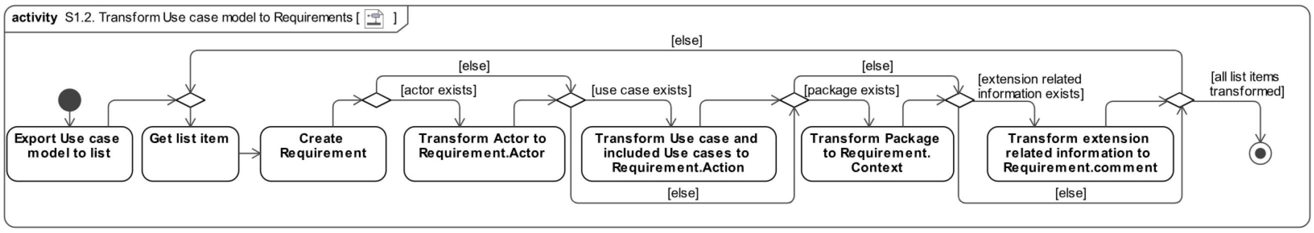

activity S1.2. Transform Use case model to Requirements
[else]
[else]
[else]
[actor exists]
[use case exists]
[package exists]
[extension related information exists]
[all list items transformed]
Export Use case model to list
Get list item
Create Requirement
Transform Actor to Requirement.Actor
Transform Use case and included Use cases to Requirement.Action
Transform Package to Requirement. Context
Transform extension related information to Requirement.comment
[else]
[else]

Figure 5. Algorithm for UML Use case model transformation into Requirements.

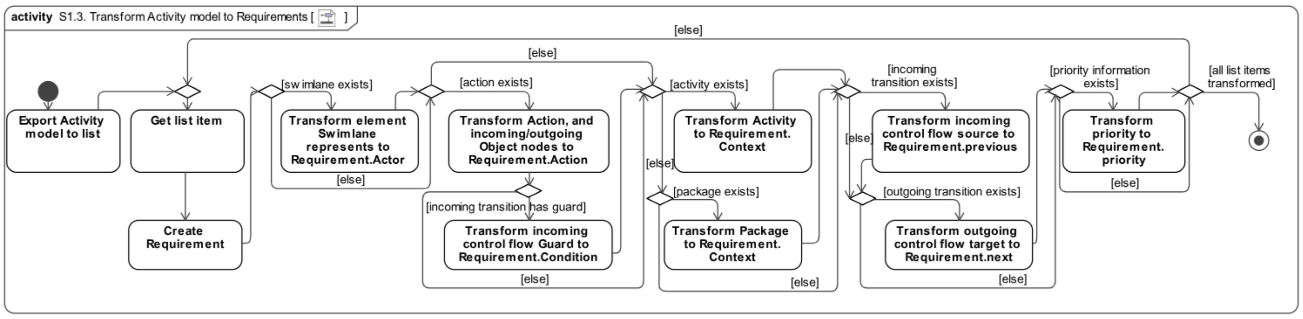

activity S1.3. Transform Activity model to Requirements
[else]
[else]
[swimlane exists]
[action exists]
[activity exists]
[incoming transition exists]
[priority information exists]
[all list items transformed]
Export Activity model to list
Get list item
Create Requirement
Transform element Swimlane represents to Requirement.Actor
[else]
Transform Action, and incoming/outgoing Object nodes to Requirement.Action
[incoming transition has guard]
Transform incoming control flow Guard to Requirement.Condition
[else]
Transform Activity to Requirement. Context
[else]
[package exists]
Transform Package to Requirement. Context
[else]
[else]
Transform incoming control flow source to Requirement.previous
[outgoing transition exists]
Transform outgoing control flow target to Requirement.next
[else]
Transform priority to Requirement. priority
[else]

Figure 6. Algorithm for UML Activity model transformation into Requirements.

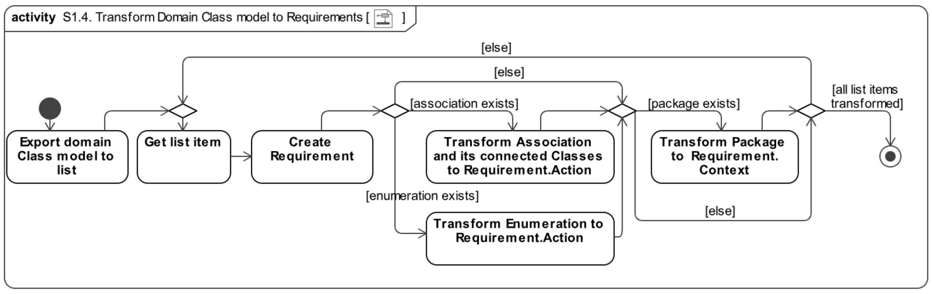

activity S1.4. Transform Domain Class model to Requirements
[else]
[else]
[association exists]
[package exists]
[all list items transformed]
Export domain Class model to list
Get list item
Create Requirement
Transform Association and its connected Classes to Requirement.Action
Transform Package to Requirement. Context
[enumeration exists]
Transform Enumeration to Requirement.Action
[else]

Figure 7. Algorithm for UML domain Class model transformation into Requirements.

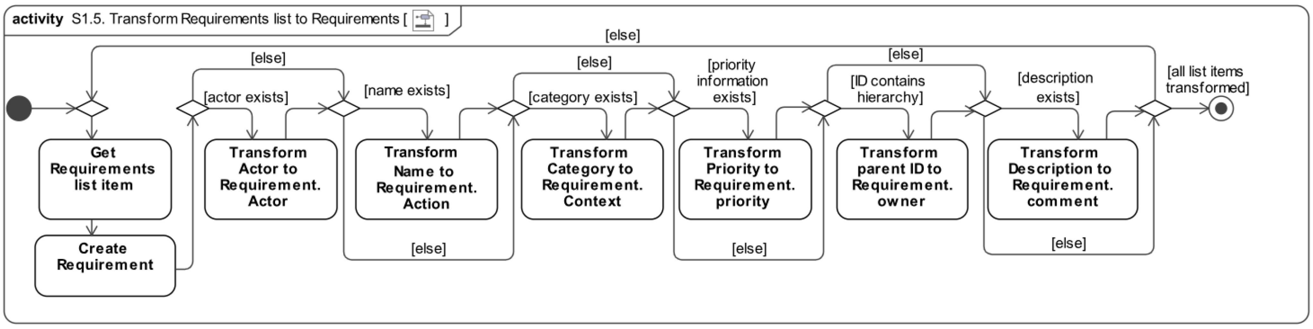

activity S1.5. Transform Requirements list to Requirements
[else]
[else]
[else]
[else]
[actor exists]
[name exists]
[category exists]
[priority information exists]
[ID contains hierarchy]
[description exists]
[all list items transformed]
Get Requirements list item
Create Requirement
Transform Actor to Requirement. Actor
Transform Name to Requirement. Action
Transform Category to Requirement. Context
Transform Priority to Requirement. priority
Transform parent ID to Requirement. owner
Transform Description to Requirement. comment
[else]
[else]
[else]

Figure 8. Algorithm for Requirements list transformation into Requirements.

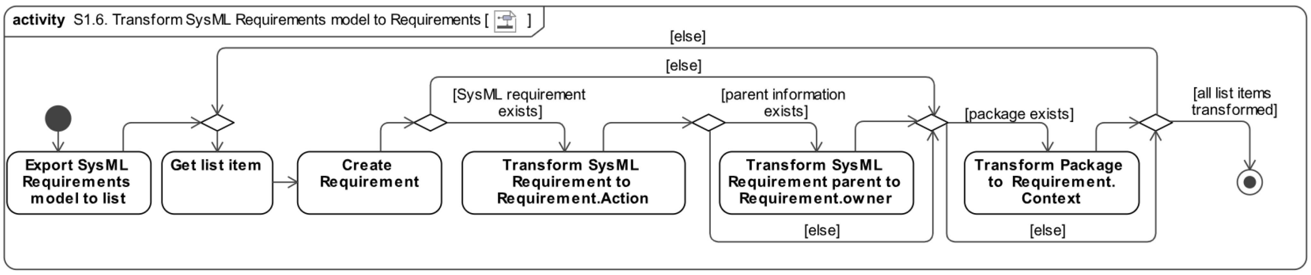

activity S1.6. Transform SysML Requirements model to Requirements
[else]
[else]
[SysML requirement exists]
[parent information exists]
[package exists]
[all list items transformed]
Export SysML Requirements model to list
Get list item
Create Requirement
Transform SysML Requirement to Requirement.Action
Transform SysML Requirement parent to Requirement.owner
Transform Package to Requirement. Context
[else]
[else]

Figure 9. Algorithm for SysML requirements model transformation into Requirements.

The Story map transformation to Requirements is presented in Figure 4. The Story map is analyzed as a text input, which contains information about Users, User stories, and Epics. Additionally, User story may have priority information, which is later stored in Requirement attribute. During the transformation, the input is exported to the list, and the list elements are analyzed one by one, transforming them into Requirement elements and attributes: Actor, Action, Context, and priority attribute.

The UML Use case model is transformed into Requirements according to the algorithm in Figure 5. The algorithm starts with exporting the model input into the list, which is further analyzed. While forming the list, for each actor, Use cases connected to it with Association are analyzed. Each Use case with its related Actor and its containing Package (if it exists) are appended to the list as an item. Use cases that are not connected with an Actor, are also analyzed and added to the list, but such list item has only the Use case element and (if exists) the Package element. Use case model Packages that do not have included Use cases are also added to the list as separate items, but these items do not have Actor and Use case elements. If an include relationship exists between the Use cases, only the including Use case is appended to the list as a separate item, but then the list item Use case part (which is further transformed into the Requirement Action) is formulated as follows: "<including Use case.name> including <included Use case.name(s)>". If an Extend relationship exists between the Use cases, both the extending and the extended Use cases are appended to list as separate items, and the extension related information is incorporated into the list item as a comment, formulated as follows: for the extended Use case "is extended by <extending Use case.name>"; for the extending Use case "extends <extended Use case.name>".

The transformation of the UML Activity model into Requirements (Figure 6) also starts with exporting the model information to the list and further analyzing the list items consecutively. Any type of Action from the Activity model is appended to the list, but initial and final Activity nodes are not analyzed. Information about Action sequence is extracted from incoming and outgoing Control Flows for each Action and is included in the Requirement attributes as previous and next. If the Action has incoming or outgoing Object flow, it must be connected to the Object node. The Object node name is treated as Object node type. If the list item (Activity model Action) has incoming Object Flow, the Requirement Action text is formed as follows: "<Action> with input(s): <Object Node.name>". If the Activity model Action has an outgoing Object Flow, the Requirement Action text is formed as follows: "<Action> with result(s): <Object Node.name>". If the Action is in the Activity Partition (Swimlane), the element, which represents the Swimlane is transformed to the Actor in the Requirement. The Guard of the incoming Control Flow of the Action is used as a Condition element of the Requirement. Action may have priority and sequence information that is stored in the attributes of the Requirement. The Activity or the Package, where the Action is contained, is transformed into the context of Requirement, but if both Package and Activity are present, Activity is used to denote the Context.

The UML Class model can be used as a source of Requirements if it is used for modeling domain concepts and their relationship. The transformation of the UML Class model to Requirements is presented in Figure 7. While forming the list from the Class model, Associations between Classes are appended to the list and then transformed into Requirements. These Requirements do not have an Actor element, only Action and Context elements. Classes that are not connected with Associations are not analyzed, but Enumerations are included, even if they are not connected with Associations. For each Association and its two connected Classes C1 and C2, a Requirement is created, where the list item part further transformed into Action is formed as follows: "<C1.name> <Association.name> <C2.AssociationEnd.Multiplicity> <C2.name>(s)". If the name of the Association is not defined, the word "has" is used instead. The reading direction of the Association is from the lesser multiplicity number to the greater or by the Association direction if it exists. If the Association type is aggregation or composition, the Action text is formed as follows: "<C1.name> consists of <C2.AssociationEnd.Multiplicity> <C2.name>(s)", where C1 is the

Class with the aggregation or composition end of Association. If Enumerations exist in the source model, they are appended to the list as separate items. The Action part of this item is formed using the Enumeration and its literals: "<Enumeration.name> can have values: <enumeration literals>".

The transformation of the Requirements list to Requirements is presented in Figure 8. Requirements list is a simple textual list, it does not need to be exported and is analyzed as is, going through all its items consecutively. The Requirements list item may have Priority, which is transformed to the Requirement priority attribute and description that is transformed to the comment attribute of the Requirement. A hierarchy in the Requirements list is stored in the ID of each list item. If the element ID has a dot (as a hierarchical separator), hierarchy is detected, and the list item, whose ID is the same as the number before the dot, is identified as a parental element (Requirement owner).

SysML Requirement model transformation to Requirements is presented in Figure 9. This transformation produces Requirements without Actor element, only Action and Context are included. Additionally, SysML Requirement may have priority and hierarchy information (the owner is identified as parental element), which are transformed to Requirement attributes.

When all source models are transformed and any required additional Requirements are appended manually, the obtained set of Requirements is further analyzed in the next steps of the scope estimation process.

3.2.2. Reconciliation of Requirements

After the initial transformation, reconciliation of requirements is the next step in achieving consensus on scope. Reconciliation of requirements consists of two steps:

- Unification of domain concepts (step S3 in Figure 2);
- Identification of overlapping requirements (Step S4 in Figure 2);

In various models prepared by different participants of requirements workshop, the same elements of domain can be named differently (e.g., user, person, and client can be used to denote the same role; furthermore, abbreviations can also be used). The unification of the domain concepts encompasses the identification of synonyms of nouns and verbs and mapping them, by deciding which one of the synonyms should be used and replacing all other synonyms with the chosen one. Identifying synonyms ensures that the terms that are used in the project are understood in the same way by everyone. The initial identification of synonyms can be performed straightforwardly according to the English dictionary, but the final mapping decisions should be made by a subject matter expert. Noun mapping is more straightforward than that of verbs, as while replacing verbs, one must ensure that they do not change the context, and each replacement must be reviewed. Thus, verb alignment takes longer and may yield less value. However, understanding the verbs used and related terminology is essential. Another important aspect is identification of similar Contexts and aggregation of these Contexts to ensure better grouping of requirements for the next step. In general, unifying the domain concepts is important not only in reconciling the requirements, but also in the formation of user wireframes and high-fidelity frames, as the same vocabulary is used. Maintaining consistency and compliance with organizational terms is one of the most important criteria for a good user experience.

The higher the number of participants in modeling requirements, the more likely it is that there will be more changes required due to synonyms used. At the same time, it ensures that the management of overlapping requirements will be more effective. Overlapping requirements often lead to duplication of effort or resources, inconsistencies, and conflicts during the project, and risks of major scope changes. Therefore, it is important to identify overlaps as early as possible in the project for ensuring consistency and quality of the requirements, clarifying the scope, and thus facilitating time efficiency, cost reduction, resource optimization, and stakeholder satisfaction. Managing of overlapping requirements consists of these steps:

- Determining the primary source (model of textual input);

- Grouping elements (optional) and identifying overlapping requirements.

Determining the primary model identifies which model the team considers to be the central or the most informative model. If it is difficult to identify, it can be determined by detecting which model outclasses others (i.e., was the source of the biggest number of generated requirements, and these requirements had the fewest replacements during unification of domain concepts).

The next step is the grouping of requirements, which facilitates reducing the set of requirements in which we search for overlaps. For grouping, two parts of requirement can be used: Actor and Context. Context grouping helps to limit the set of requirements for identifying overlaps by area of functionality, as there is no need to compare requirements that describe different areas (e.g., registration or order creation). Actor grouping facilitates limiting the set of requirements in terms of roles responsible for the certain requirements, as it is easier to identify the overlapping functionality specified for the same role. Both Actor and Context grouping strategies are useful, but we suggest applying each of them consecutively to gain the maximum benefits and ensure thorough identification of overlaps.

Identification of overlapping requirements is performed by calculating the degree of overlap expressed as a percentage, which depends on which parts of the requirement overlap. As there are four main parts in the requirement template (Actor, Action, Condition, and Context), overlapping can occur in any of these parts and their combinations. The analysis of the Action part constitutes the main weight in calculating the degree of overlap. If the verb in the Action part is identified as overlapping with the Action part of the other requirement, a 50% match is calculated. If the rest of the requirement parts (Actor, Condition, Context) match, that makes up the remaining 50%, dividing it for each requirements part accordingly. Each requirement can be assigned as overlapping with one or several other requirements. If the degree of overlap is less than 100% but greater than or equal to 50%, the overlap should be approved or rejected by the domain expert. Usually, if the degree of overlap is less than 50, it is ignored, unless the margin of ignoring the overlap is chosen to be smaller. After this phase, the team members have a list of requirements that reflects their opinion and knowledge of the subject area and scope of the project.

The overlapping requirements are then removed from the list, leaving only one of them. When requirements from different sources are identified as overlapping, the decision must be made as to which one of the requirements must be removed. Additionally, the alignment of the sequence of requirements can be useful for the identification of overlaps. It involves sequencing the requirements in one context. The requirements transformed from UML Activity model are perfect for this, and can be supplemented by requirements from other models, requiring them to be sequenced. This is not a mandatory step, but it allows one to determine that there are no missed or redundant requirements in the process. Additional information such as non-functional requirements, technical, architecture requirements, or risks can be added to the requirement set as well. These do not directly reflect functionality; their existence may increase project costs and alter the schedule and the required number of resources; therefore, they should be incorporated into the requirements list.

The reconciliation process has its challenges related to both the human-related aspects of team decision-making processes and to the technical side of identifying overlaps and synonyms. As domain understanding can vary between team members and the time allocated to domain analysis is extremely limited, conflicts can arise in the team while realigning requirements and deciding on the most relevant domain vocabulary terms. These conflicts require additional time for discussions and reaching a consensus within the team. From a more technical perspective, while the tool supports basic text analysis, the reconciliation of requirements mainly relies on individual expertise of the team members, and it requires an in-depth text analysis. At the same time, the analysis must also be performed under time pressure, which may hinder identification of the overlaps that are not apparent at first glance.

3.3. Scope Estimation

After collecting and reconciling the requirements, the project scope estimation is performed (step S5 in Figure 2), which encompasses defining the priorities and level of clarity, evaluating the story points, and calculating the estimate for each requirement. This is performed by reaching a group consensus, according to the story point estimation principles [37].

Prioritization is an optional step, but it can be useful for calculating the scope of various requirement sets, grouped by priority. Depending on the information available, priorities can be obtained from the source models or introduced manually. If necessary (or when the team decides to do so), the priority of the requirement can be adjusted, or the requirement can be removed from the scope assessment. This is the initial prioritization. More detailed prioritization is performed later during the development process.

Additionally, a level of clarity is identified for each requirement. We suggest using four levels of clarity (although these levels can be modified according to the specifics of the project and the team):

- Level 1 (the requirement is very clear, and its purpose is understandable);
- Level 2 (the requirement raises questions and is not completely clear);
- Level 3 (the requirement has several solution options, it is not clear which way it will have to be carried out);
- Level 4 (the requirement is not clear, which raises many questions that will be answered after additional analysis or implementation of other functionalities).

The coefficient of clarity for each level can be custom; we suggest using 1 for level 1, 1.3 for level 2, 1.5 for level 3, and 2 for level 4. The coefficient of clarity increases the amount of work that will potentially be required for the requirement.

The next step is using expert judgement to assess the scope of each requirement in story points according to Agile estimation principles [37]. If the scope of the requirement is very small, the requirement can be merged with other requirements to make the list clearer and easier to control. If the scope of the requirement is very large, it should be broken down to make its components clearer. After assessing all requirements, the scope can be estimated for each requirement according to the following formula:

$$\textit{Requirement Scope} = \textit{Requirement Story Points} * \textit{Coefficient of Clarity} \tag{1}$$

The distribution of requirements according to the assigned level of clarity and the number of story points assessed is a good indicator of the quality of the results of requirements elicitation. If the requirements are mostly evaluated in 3–8 story points and assigned level 1 or level 2 coefficients of clarity, it can be concluded that a manageable list of requirements is obtained, and it can be used in the next steps of the project. In the opposite case of the predominance, i.e., when requirements are evaluated in 13 and more story points are level 3 or level 4, coefficients of clarity prevail. In this case, it is necessary to either repeat the workshop or offer the client an additional requirements discovery phase, during which the requirements would be detailed, and their complexity and level of uncertainty would be reduced. The level of clarity of each requirement group (if requirement grouping is used) is also important. This helps to understand whether it will be possible to start development as soon as the project starts or if it is needed to pay more attention to the analysis. If there are no requirement groups where the clarity is good (level 1 or level 2), then the start of the project will be complicated, with limitations in parallel implementation of several parts of the system under development.

Finally, the estimate of all project scope is acquired by summing up all calculated requirements scopes. Other strategies for estimation can also be used, e.g., including only requirements with high and very high priorities in project scope estimation.

4. Implementation of the Proposed Methodology

The proposed methodology was implemented in a PSES tool (Project Scope Estimation System), which performs the main steps of the proposed process. In the first step, the user can import the source models or other textual sources. The tool imports .xmi files to gather requirements from the UML and SysML models and .csv files to gather requirements from the requirements list or Story map. The tool analyzes the imported information and generates a list of requirements. The screenshots captured while using the tool in the animal shelter software development project with the imported list of requirements are presented in Section 5. The user can see from which model the data came. Also, the structure of the requirement is represented by coloring, which facilitates easier identification of the Actor, Action, and Context parts. In this way, the tool enables the user to quickly search for and organize it. During transformation, the requirement receives a unique number. The list contains the requirement text, priority, owner, dependency sequence, and model number from which the requirement was retrieved.

In the next step, users can analyze and replace synonyms in the requirements. The identification of overlaps is also partially automated using the tool, but the final decisions about overlap confirmation must be made by the user. The final processing of the requirements is performed in the next step, after which the scope estimation is carried out. During scope estimation, the user must enter the story points for each requirement in the list, along with the clarity coefficients, and the tool calculates the final estimate for each task and for the entire project.

5. Case Study of Applying the Proposed Scope Estimation Methodology

The early phase project scope estimation methodology was applied in the information system development project performed in an international IT company department based in Lithuania. The IT company works on various software product development projects.

The project analyzed in this case study started in 2022, and a minimum viable product (MVP) was developed. The project subject area is the pet shelter "One Tail At A Time (OTAAT)". The goal of the shelter is to end pet homelessness by making pet ownership a joyful and accessible experience for all. This is achieved by rescuing animals from overcrowded shelters, placing them in loving forever homes, and providing support and resources to pet owners in need. The pet adoption process has historically been outdated, and OTAAT is looking to tackle that via the digital space. They want to better manage applications, as well as quickly match and place pets into homes. In the existing process, people call the office to inquire about pet availability and schedule visits. Applications are downloaded, printed, filled out, and then sent back to the Adoption Center Director. The specialized software can help in optimizing and streamlining existing processes.

After the requirements workshop, the initial requirements were gathered using Story map (prepared using Miro and exported as CSV file), UML Use case, UML domain Class, and UML Activity models (created using Magic Systems of Systems Architect and exported as XMI file). The requirements were gathered by the product manager, the product designer, the architect, the business analyst, and the technical delivery manager, in cooperation with several participants from the OTAAT side.

One Story map model was created by the team working together with the client during the requirements workshop. Team members also prepared three Use case models (by product manager, business analyst, and architect) and nine Activity models (by product designer, business analyst, and technical delivery manager) that served as detailed information in addition to the Story map. There was only one domain Class model created by the architect. The fragments of the prepared models are presented in Figures 10–12.

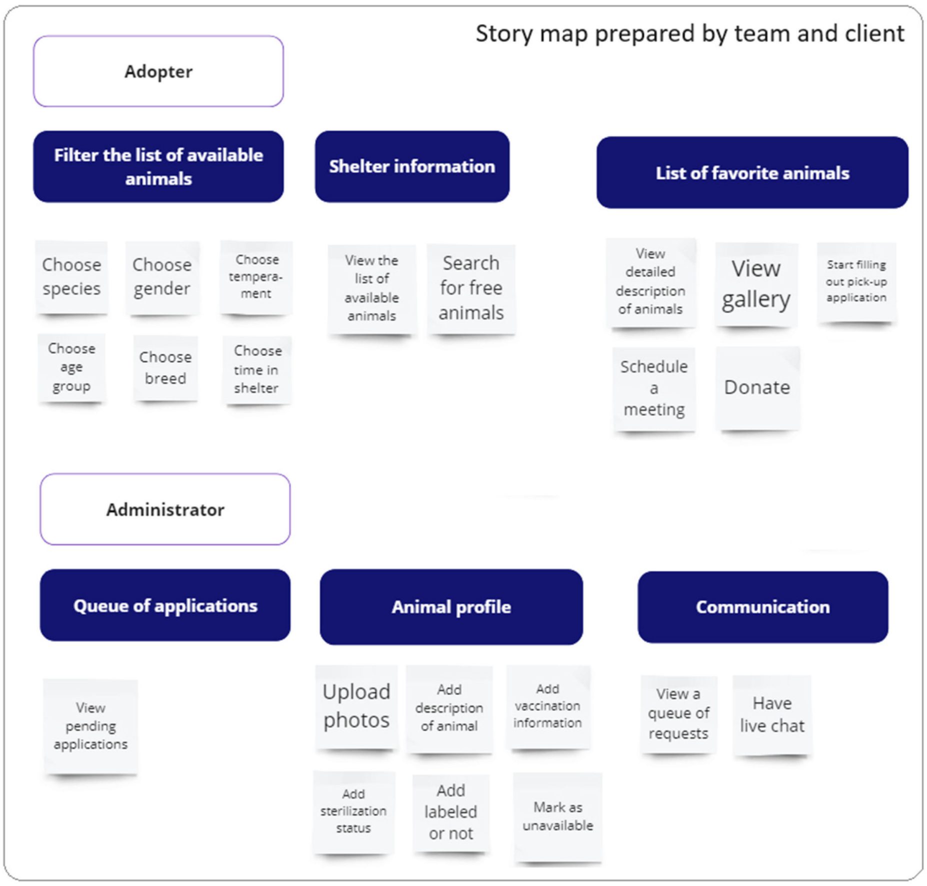


Figure 10. Fragment of the Story map for OTAAT project prepared during the requirements workshop.

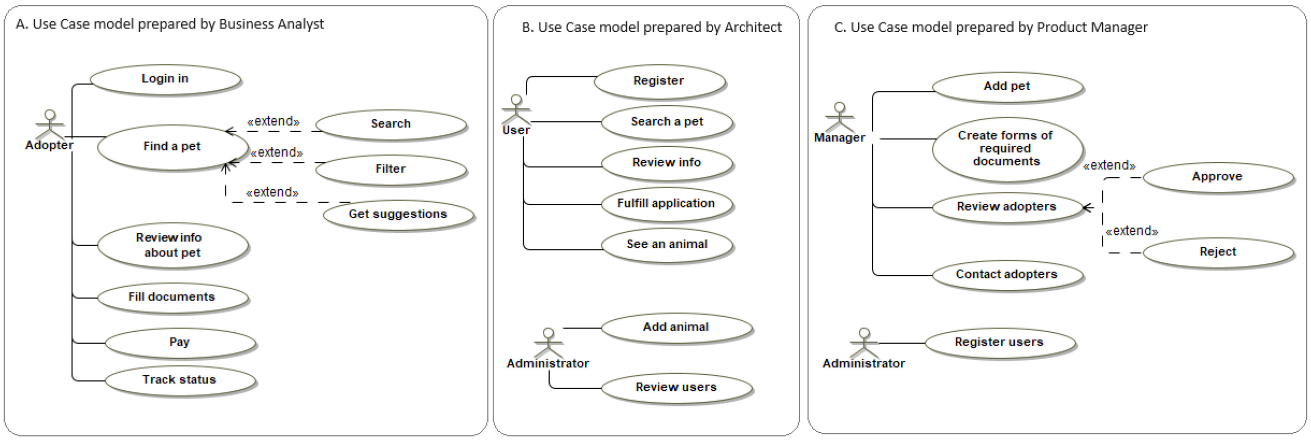


Figure 11. Examples of UML Use case models for OTAAT project prepared during the requirements workshop.

The prepared models have a lot in common, such as repeating users and similar action names. However, performing rapid manual model comparison and aggregation is difficult, time consuming, and inefficient. For this reason, the tool was used to import the source models and generate a list of requirements (Figure 13). Information about the number of requirements gathered from all prepared models is presented in Table 2.

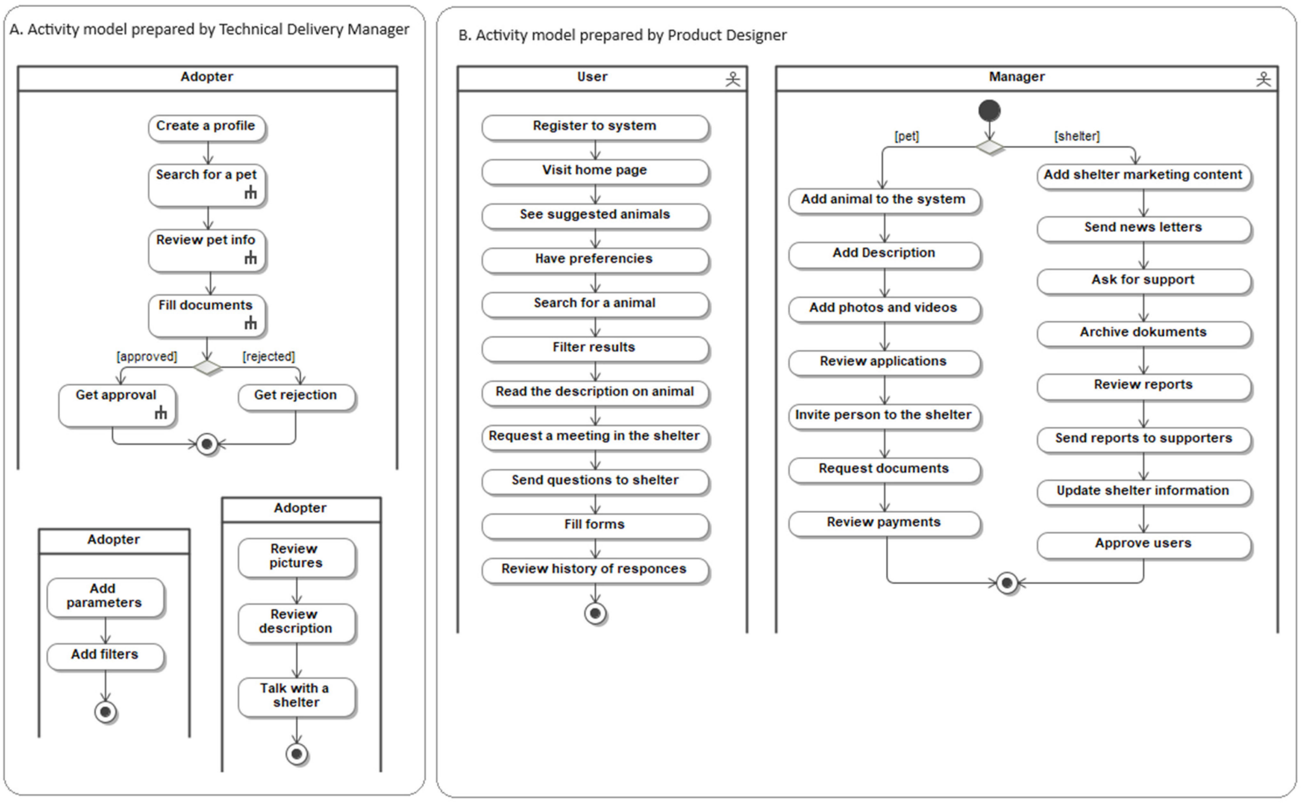


Figure 12. Examples of UML Activity models for OTAAT project prepared during the requirements workshop.

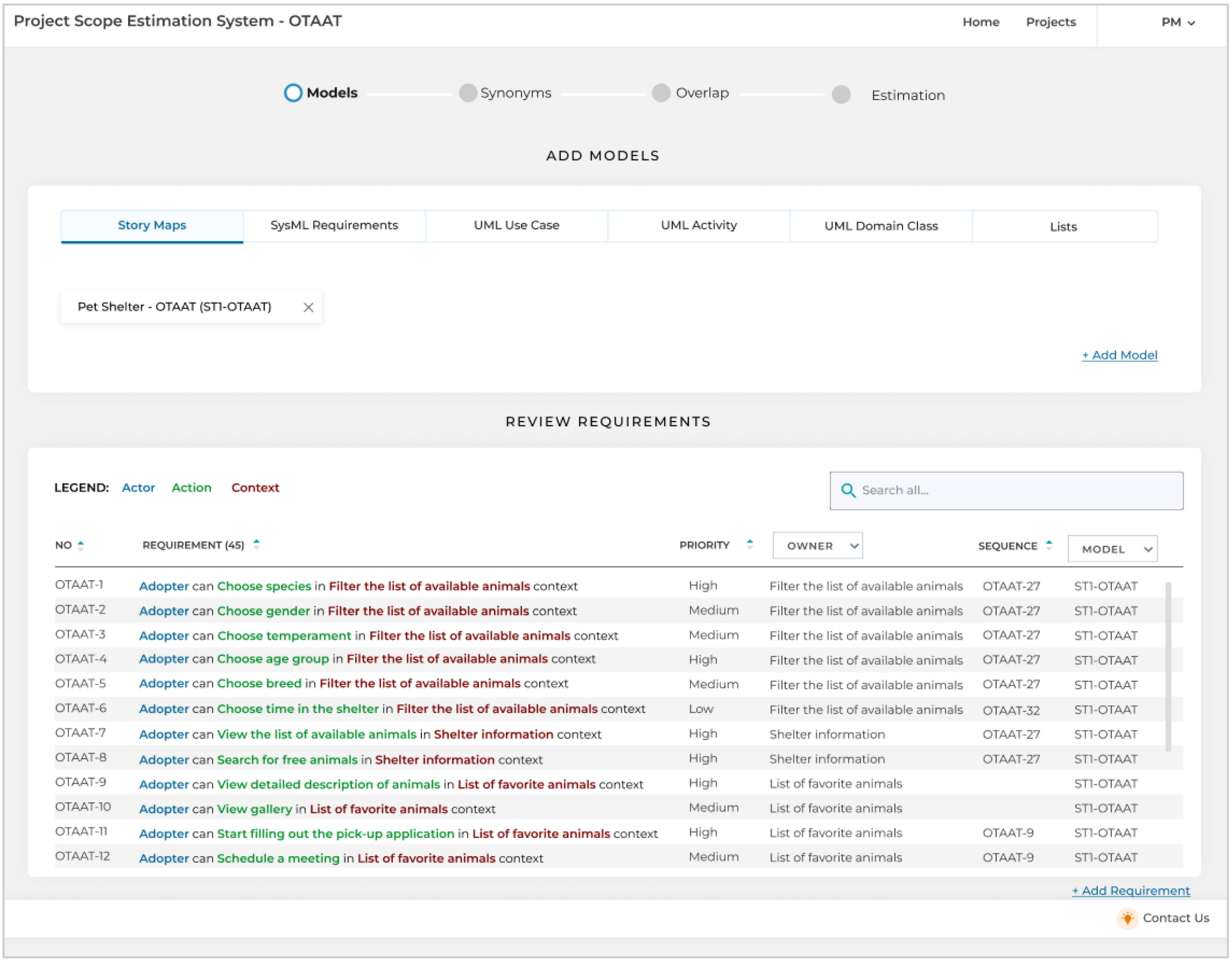
In the next step, the synonyms were analyzed (Step S3 in Figure 2). Two overlapping actors, the Administrator and the Manager, were found. After unification, the Administrator in all requirements was renamed to Manager. In total, there were 19 changes. The naming of the Adopter actor was also unified, as several team members named it User (thus, User was changed to Adopter in 20 cases). Other overlapping nouns were also found and replaced, such as animal to pet (44 replacements); organization to shelter (1 replacement); confirmation to approval (1 replacement); registration to login (3 replacements); and document to form (9 replacements). In total, 97 changes were made during the unification of noun domain concepts.

The unification of verbs also revealed usage of synonyms (e.g., add and create, search and find, ask and request, approve and confirm), but the number of replaced synonyms in verbs (in total, 18 replacements) was smaller than that of nouns. This is quite common, as in software development, the names of actions are typical, and development teams more often use verbs purposefully in the same way. On the other hand, nouns depend mostly on the domain of the product and can differ significantly between the projects, thus leading to the occurrence of a larger number of noun synonyms not carrying a distinct meaning.

Analysis of synonyms enabled identification of similar and/or overlapping Contexts. Team members aggregated contexts that had similar meaning. Before analysis, the requirements list had 44 contexts, and, after aggregation, the list contained 13 contexts. Examples of aggregated contexts are “Search pet” and “Search for a pet”, “Pet” and “Pet profile”, “Add pet to system” and “Add pet”.

After the analysis of synonyms, the identification of overlapping requirements was performed (Step S4 in Figure 2). The previously performed unification of terms enabled more accurate identification of requirement overlaps. In the OTAAT project, the Story map was chosen as the primary model. The grouping of requirements was carried out by Actor, also filtering the requirements by Context. During the overlap identification step, 11 requirements from Story Map model, 14 requirements from Use case models, 1 from

Class model, and 16 requirements from Activity models (in total 42 requirements) were identified as overlapping and were removed. For example, the requirements "Manager can Review adopters in Application context" (from the Use case model), "Manager can Review applications in Application context" (from another Use case model), and "Manager can Review adopters information in Application context" (from the Activity model) were identified as overlapping, and after team discussion, the last one of these requirements was selected as suitable and the other two were removed. In another example, the requirements "Adopter can Schedule a meeting in Pet profile context" (from Story map) and "Adopter can Request a meeting in the shelter in Pet profile context" (from the Activity model) were identified as overlapping, and the second one was removed as duplicate.



NO	REQUIREMENT (45)	PRIORITY	OWNER	SEQUENCE	MODEL
OTAAT-1	Adopter can Choose species in Filter the list of available animals context	High	Filter the list of available animals	OTAAT-27	ST1-OTAAT
OTAAT-2	Adopter can Choose gender in Filter the list of available animals context	Medium	Filter the list of available animals	OTAAT-27	ST1-OTAAT
OTAAT-3	Adopter can Choose temperament in Filter the list of available animals context	Medium	Filter the list of available animals	OTAAT-27	ST1-OTAAT
OTAAT-4	Adopter can Choose age group in Filter the list of available animals context	High	Filter the list of available animals	OTAAT-27	ST1-OTAAT
OTAAT-5	Adopter can Choose breed in Filter the list of available animals context	Medium	Filter the list of available animals	OTAAT-27	ST1-OTAAT
OTAAT-6	Adopter can Choose time in the shelter in Filter the list of available animals context	Low	Filter the list of available animals	OTAAT-32	ST1-OTAAT
OTAAT-7	Adopter can View the list of available animals in Shelter information context	High	Shelter information	OTAAT-27	ST1-OTAAT
OTAAT-8	Adopter can Search for free animals in Shelter information context	High	Shelter information	OTAAT-27	ST1-OTAAT
OTAAT-9	Adopter can View detailed description of animals in List of favorite animals context	High	List of favorite animals		ST1-OTAAT
OTAAT-10	Adopter can View gallery in List of favorite animals context	Medium	List of favorite animals		ST1-OTAAT
OTAAT-11	Adopter can Start filling out the pick-up application in List of favorite animals context	High	List of favorite animals	OTAAT-9	ST1-OTAAT
OTAAT-12	Adopter can Schedule a meeting in List of favorite animals context	Medium	List of favorite animals	OTAAT-9	ST1-OTAAT

Figure 13. Using the estimation tool in the OTAAT project to import information from source models.

Table 2. Numbers of requirements extracted from various source models.

Source Model Type	Number of Models Prepared by Team and Used in Transformation	Number of Requirements Transformed from the Source
Story map	1	47
UML Use case	3	22
UML Class	1	6
UML Activity	9	58

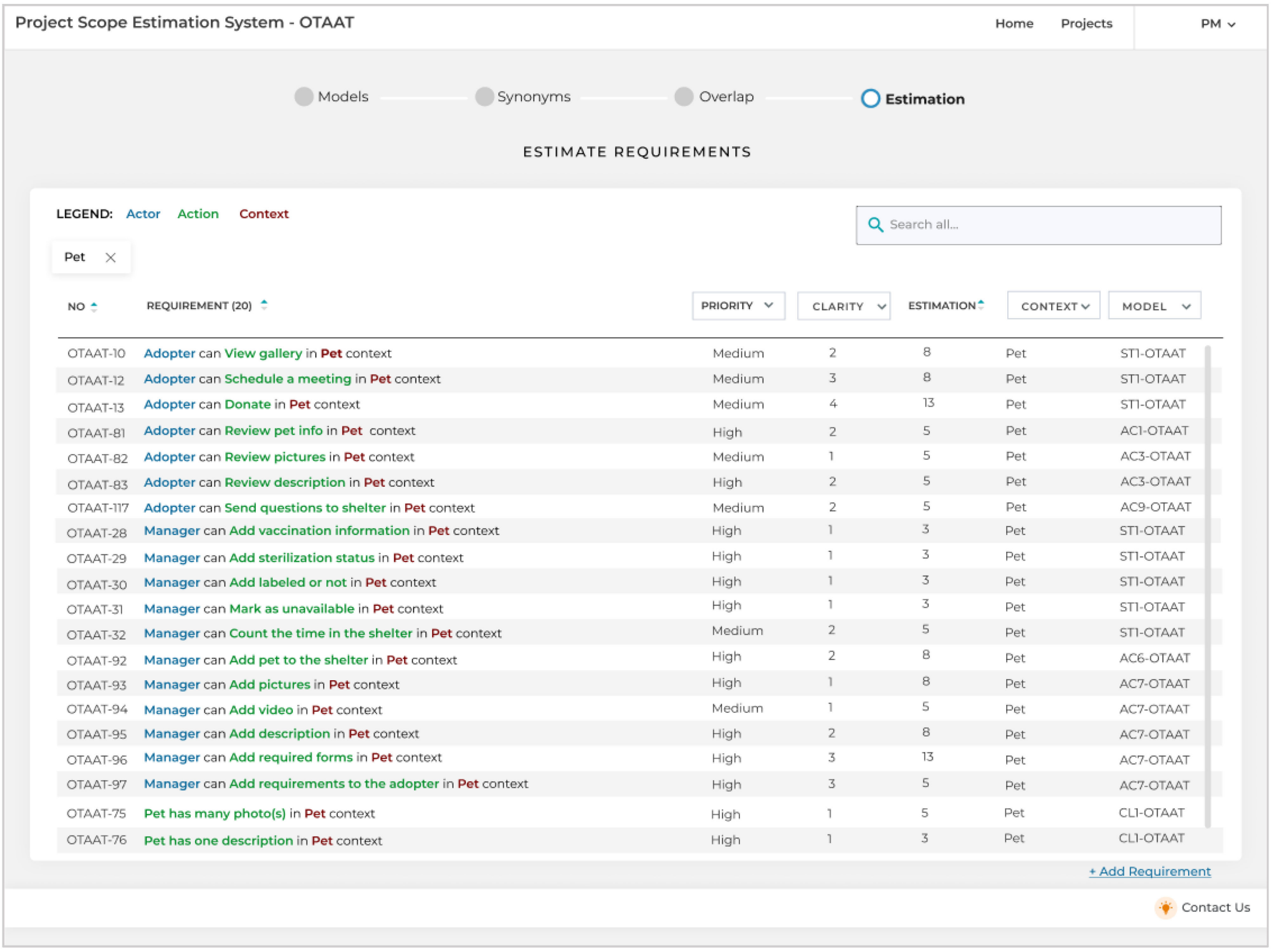
The final list of requirements contained 91 requirements, of which 36 were transformed from the Story map, 8 were gathered from Use case models, 5 came from the Class model, and 42 were gathered from Activity models. Detailed results of the requirement transformation from different sources and reconciliation are presented in Table 3.

Table 3. Numbers of requirements after transformation from source models and reconciliation (NoRAT—number of requirements after transformation from source; NoRO—number of requirements removed as overlapping; FNoR—final number of requirements after reconciliation).

Context	**NoRAT**	**Transformed from Story Map**		**Transformed from Use Case Model**		**Transformed from Class Model**		**Transformed from Activity Model**		**FnoR**
		NoRAT	**NoRO**	**NoRAT**	**NoRO**	**NoRAT**	**NoRO**	**NoRAT**	**NoRO**	
					Actor: Adopter					
Search for a pet	17	6	0	6	4	0	0	5	2	11
Shelter information	2	2	1	0	0	0	0	0	0	1
Pet	14	4	1	1	1	2	0	7	3	9
Application	25	9	3	5	3	2	0	9	1	18
Communication	1	1	0	0	0	0	0	0	0	1
Login	10	6	2	2	1	0	0	2	1	6
Adopter profile	4	0	0	0	0	1	0	3	0	4
Archive	3	3	0	0	0	0	0	0	0	3
Total	76	31	7	14	9	5	0	26	7	53
					Actor: Manager					
Pet	19	7	2	2	2	1	1	9	3	11
Adopter profile	4	2	0	1	0	0	0	1	0	4
Application	21	1	1	5	3	0	0	15	5	12
Archive	9	6	1	0	0	0	0	3	1	7
Shelter	4	0	0	0	0	0	0	4	0	4
Total	57	16	4	8	5	1	1	32	9	38

The next and final step was the scope estimation (Step S5 in Figure 2). During this step, the whole team worked together, discussing each requirement. The requirement was given a level of clarity and an estimate of story points. The fragment of the requirements list with the provided estimates is visible in the screenshot of the scope estimation tool usage during the estimation step (Figure 14).

The distribution of requirements according to the level of clarity and the number of story points is presented in Figure 15. The highest number of requirements is seen in the 3–5–8 story point categories and clarity levels 1 and 2, with a total of 61, which indicates a satisfactory status of requirements elicitation. The requirements are manageable, meaning not too large in story points, and a more detailed analysis is not required for most of the requirements.


Project Scope Estimation System - OTAAT

Home Projects PM

Models — Synonyms — Overlap — Estimation

ESTIMATE REQUIREMENTS

LEGEND: Actor Action Context

Search all...

Pet

NO	REQUIREMENT (20)	PRIORITY	CLARITY	ESTIMATION	CONTEXT	MODEL
OTAAT-10	Adopter can View gallery in Pet context	Medium	2	8	Pet	ST1-OTAAT
OTAAT-12	Adopter can Schedule a meeting in Pet context	Medium	3	8	Pet	ST1-OTAAT
OTAAT-13	Adopter can Donate in Pet context	Medium	4	13	Pet	ST1-OTAAT
OTAAT-81	Adopter can Review pet info in Pet context	High	2	5	Pet	AC1-OTAAT
OTAAT-82	Adopter can Review pictures in Pet context	Medium	1	5	Pet	AC3-OTAAT
OTAAT-83	Adopter can Review description in Pet context	High	2	5	Pet	AC3-OTAAT
OTAAT-117	Adopter can Send questions to shelter in Pet context	Medium	2	5	Pet	AC9-OTAAT
OTAAT-28	Manager can Add vaccination information in Pet context	High	1	3	Pet	ST1-OTAAT
OTAAT-29	Manager can Add sterilization status in Pet context	High	1	3	Pet	ST1-OTAAT
OTAAT-30	Manager can Add labeled or not in Pet context	High	1	3	Pet	ST1-OTAAT
OTAAT-31	Manager can Mark as unavailable in Pet context	High	1	3	Pet	ST1-OTAAT
OTAAT-32	Manager can Count the time in the shelter in Pet context	Medium	2	5	Pet	ST1-OTAAT
OTAAT-92	Manager can Add pet to the shelter in Pet context	High	2	8	Pet	AC6-OTAAT
OTAAT-93	Manager can Add pictures in Pet context	High	1	8	Pet	AC7-OTAAT
OTAAT-94	Manager can Add video in Pet context	Medium	1	5	Pet	AC7-OTAAT
OTAAT-95	Manager can Add description in Pet context	High	2	8	Pet	AC7-OTAAT
OTAAT-96	Manager can Add required forms in Pet context	High	3	13	Pet	AC7-OTAAT
OTAAT-97	Manager can Add requirements to the adopter in Pet context	High	3	5	Pet	AC7-OTAAT
OTAAT-75	Pet has many photo(s) in Pet context	High	1	5	Pet	CL1-OTAAT
OTAAT-76	Pet has one description in Pet context	High	1	3	Pet	CL1-OTAAT

+ Add Requirement

Contact Us

Figure 14. Using the estimation tool in the OTAAT project Estimation step.

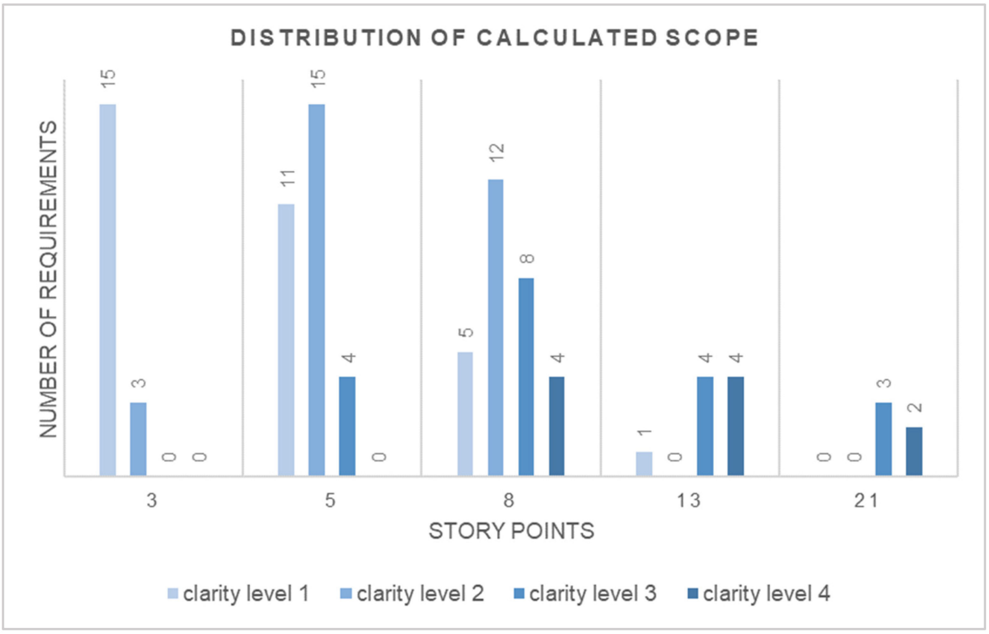


Figure 15. The results of the scope estimation for the OTAAT requirements in terms of story points and levels of clarity.

If the OTAAT project scope estimation had been based only on the Story map (as is quite common in Agile software development projects), the number of gathered requirements would have been much smaller, as only 47 requirements were captured in the Story map during the OTAAT requirements workshop. These 47 requirements were evaluated in 566 story points. By introducing additional models during the requirements workshop, the team was able to capture more of the potential product requirements, and the final number of gathered requirements was 91. The final estimate of the scope for these 91 requirements was 938 story points, which revealed a higher complexity of the potential product than the estimate based only on the Story map.

Such a strategy of estimation ensures awareness of the higher complexity of the project in its early phase, both internally in the team and by the client. For the OTAAT project, the contract was prepared for the development of minimum viable product (MVP) based only on the requirements with the highest priority. The number of highest priority requirements used to calculate the MVP scope was 50, and the calculated scope itself was 408 story points. Of the 50 highest priority requirements, 12 were captured in the Story map, and 38 originated from other source models. Some of these 38 requirements might have been also captured in the Story map, but were eliminated during the overlap analysis, when the requirement from another source model was chosen as more suitable.

The OTAAT project development team was satisfied with the proposed early phase scope estimation methodology, and stated that the usage of the methodology increased the clarity of the project requirements and revealed its complexity. It has raised the teams' confidence in the captured requirements and the calculated scope. It also helped ensure better alignment both within the team and with the client.

6. Discussion and Conclusions

We have proposed an expert-based methodology for software effort estimation, suitable for use in early phase estimation of Agile software development projects. Unlike traditional software effort estimation methods, our methodology can be applied in a time-limited context of early software requirements elicitation. It also supports the elicitation of requirements from various media: Story maps, requirements lists, UML Use case models, UML Activity models, UML Class models, and SysML Requirement models. The use of various forms for the requirements elicitation allows the involvement of stakeholders with various competencies. For the project scope estimation, a single list of requirements is composed by extracting requirements from the various media used by the participants of the early requirements workshop. To avoid duplicates and contradictions, requirements are then deduplicated and reconciled with each other. The final list of requirements is used to estimate the project scope by assigning story points to the requirements via an expert-based approach.

The case study has demonstrated that using several different models for requirements elicitation allowed different aspects of solution to be discussed. While Story map was used as a central model, other models have allowed additional requirements to be contributed, which might have been missed if only Story map had been used.

The analysis of synonyms allowed noun and verb synonyms to be identified and eliminated to unify the domain concepts. It also enabled the identification of overlapping contexts of requirements and reducing the number of distinct contexts by aggregating the overlapping ones. A smaller number of distinct contexts made the final requirements list more manageable.

After identifying overlaps, the final count of requirements was reduced to 91 from 133 initially. This indicates the usefulness of automated overlap analysis, since otherwise there would have been a large amount of wasted effort produced by overlapping requirements.

The scope estimation phase has resulted in most requirements having up to 8 story points and an average level of clarity. Several requirements were assigned 13 and 21 story points, which indicates that they should be further broken down into smaller parts. However, in the early scope assessment phase, having a few "heavy" requirements is generally acceptable, because subsequent splitting could require additional time that is not available in the early requirements workshop.

The case study using the proposed methodology demonstrated that using UML Use case, Activity, and Class diagrams as additional sources of requirements in addition to the Story map can increase the number of discovered requirements to be used in project scope estimation. The possibility of recording requirements in various forms prepared by different workshop participants and then reconciling them allowed us to calculate a more detailed estimate for the demonstrated project.

In the future, we are planning to continue our research in two main directions. First, we aim to apply the methodology to a larger set of software development projects covering a wider variety of domains and project sizes, thus exploring the scalability of the methodology. Second, we also plan to improve the proposed methodology and tool by employing machine learning techniques in the requirements reconciliation step to reduce reliance on individual expertise and better support decision-making processes. Machine learning models can be helpful in supporting team members in identifying synonyms and/or requirement overlaps and aggregating similar contexts, thus further streamlining the reconciliation step.

Author Contributions: Conceptualization, L.B., L.Č. and M.J.; methodology, L.B., L.Č. and E.G.; software, E.G. and L.B.; validation, L.Č., M.J. and L.A.; formal analysis, L.B. and L.Č.; investigation, L.B.; resources, L.B. and L.Č.; writing—original draft preparation, L.B., L.Č. and M.J.; writing—review and editing, L.B., L.Č., M.J. and L.A.; visualization, L.B. and L.Č.; supervision, L.B.; project administration, L.Č. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to confidentiality of information about software development project.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hoy, Z.; Xu, M. Agile Software Requirements Engineering Challenges-Solutions—A Conceptual Framework from Systematic Literature Review. *Information* **2023**, *14*, 322. [CrossRef]
2. Kasauli, R.; Knauss, E.; Horkoff, J.; Liebel, G.; Neto, F.G. Requirements engineering challenges and practices in large-scale agile system development. *J. Syst. Softw.* **2021**, *172*, 110851. [CrossRef]
3. Ambler, S. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*; John Wiley & Sons: Hoboken, NJ, USA, 2002.
4. Alfraihi, H.; Lano, K. The Integration of Agile Development and Model Driven Development—A Systematic Literature Review. In Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development, Porto, Portugal, 19–21 February 2017; SciTePress: Setúbal, Portugal, 2017; Volume 1. MODELSWARD. [CrossRef]
5. Object Management Group. Unified Modeling Language Specification Version 2.5. 2015. Available online: https://www.omg.org/spec/UML/2.5 (accessed on 1 November 2023).
6. Object Management Group. Systems Modeling Language Specification Version 1.2. 2010. Available online: https://www.omg.org/spec/SysML/1.2 (accessed on 1 November 2023).
7. Moyano, C.G.; Pufahl, L.; Weber, I.; Mendling, J. Uses of business process modeling in agile software development projects. *Inf. Softw. Technol.* **2022**, *152*, 107028. [CrossRef]
8. Liebel, G.; Knauss, E. Aspects of modelling requirements in very-large agile systems engineering. *J. Syst. Softw.* **2023**, *199*, 111628. [CrossRef]
9. Nataliia, D.; Dmytro, C.; Igor, C. Modeling of the Processes of Stakeholder Involvement in Command Management in a Multi-Project Environment. In Proceedings of the 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT), Lviv, Ukraine, 11–14 September 2018; IEEE: Piscataway, NJ, USA, 2018. [CrossRef]
10. Buganová, K.; Šimíčková, J. Risk management in traditional and agile project management. *Transp. Res. Procedia* **2019**, *40*, 986–993. [CrossRef]

11. Singh, S.; Madaan, G.; Singh, A.; Sood, K.; Grima, S.; Rupeika-Apoga, R. The AGP Model for Risk Management in Agile I.T. Projects. *J. Risk Financ. Manag.* **2023**, *16*, 129. [CrossRef]
12. Inayat, I.; Salim, S.S.; Marczak, S.; Daneva, M.; Shamshirband, S. A systematic literature review on agile requirements engineering practices and challenges. *Comput. Hum. Behav.* **2015**, *51 Pt B*, 915–929. [CrossRef]
13. Wohlrab, R.; Knauss, E.; Pelliccione, P. Why and how to balance alignment and diversity of requirements engineering practices in automotive. *J. Syst. Softw.* **2020**, *162*, 110516. [CrossRef]
14. Trendowicz, A.; Jeffery, R. *Software Project Effort Estimation: Foundations and Best Practice Guidelines for Success*; Springer: Cham, Switzerland, 2014. [CrossRef]
15. Fernández-Diego, M.; Méndez, E.R.; González-Ladrón-De-Guevara, F.; Abrahão, S.; Insfran, E. An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review. *IEEE Access* **2020**, *8*, 166768–166800. [CrossRef]
16. Leffingwell, D. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*; Addison-Wesley Professional: Boston, MA, USA, 2010.
17. Adomavičius, A. *The Secret Source: The Culture, Skills, and Process for Making Great Digital Products*; Devbridge: Toronto, ON, Canada, 2021.
18. Saeeda, H.; Dong, J.; Wang, Y.; Abid, M.A. A proposed framework for improved software requirements elicitation process in SCRUM: Implementation by a real-life Norway-based IT project. *J. Softw. Evol. Process* **2020**, *32*, e2247. [CrossRef]
19. Dragicevic, S.; Celar, S.; Novak, L. Use of Method for Elicitation, Documentation, and Validation of Software User Requirements (MEDoV) in Agile Software Development Projects. In Proceedings of the Sixth International Conference on Computational Intelligence, Communication Systems and Networks, Tetova, Macedonia, 27–29 May 2014; IEEE: Piscataway, NJ, USA, 2014. [CrossRef]
20. De Vito, G.; Ferrucci, F.; Gravino, C. Design and automation of a COSMIC measurement procedure based on UML models. *Softw. Syst. Model.* **2020**, *19*, 171–198. [CrossRef]
21. Hacaloglu, T.; Demirors, O. Measureability of Functional Size in Agile Software Projects: Multiple Case Studies with COSMIC FSM. In Proceedings of the 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Kallithea, Greece, 28–30 August 2019; IEEE: Piscataway, NJ, USA, 2019. [CrossRef]
22. 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. [CrossRef]
23. Rosa, W.; Madachy, R.; Clark, B. Early Phase Cost Models for Agile Software Processes in the US DoD. In Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Toronto, ON, Canada, 9–10 November 2017; IEEE: Piscataway, NJ, USA, 2017. [CrossRef]
24. Rosa, W.; Clark, B.K.; Madachy, R.; Boehm, B.W. Empirical Effort and Schedule Estimation Models for Agile Processes in the US DoD. *IEEE Trans. Softw. Eng.* **2021**, *48*, 3117–3130. [CrossRef]
25. Rodríguez Sánchez, E.; Vázquez Santacruz, E.F.; Cervantes Maceda, H. Effort and Cost Estimation Using Decision Tree Techniques and Story Points in Agile Software Development. *Mathematics* **2023**, *11*, 1477. [CrossRef]
26. Fu, M.; Tantithamthavorn, C. GPT2SP: A Transformer-Based Agile Story Point Estimation Approach. *IEEE Trans. Softw. Eng.* **2023**, *49*, 611–625. [CrossRef]
27. Tanveer, B.; Guzmán, L.; Engel, U.M. Effort estimation in agile software development: Case study and improvement framework. *J. Softw. Evol. Process* **2017**, *29*, e1862. [CrossRef]
28. Tukur, M.; Umar, S.; Hassine, J. Requirement Engineering Challenges: A Systematic Mapping Study on the Academic and the Industrial Perspective. *Arab. J. Sci. Eng.* **2021**, *46*, 3723–3748. [CrossRef]
29. Boggero, L.; Ciampa, P.D.; Nagel, B. An MBSE architectural framework for the agile definition of system stakeholders, needs and requirements. In Proceedings of the AIAA Aviation 2021 Forum, Virtual, 2–6 August 2021; American Institute of Aeronautics and Astronautics: Reston, VA, USA, 2021. [CrossRef]
30. Shukla, S.; Kumar, S. Know-UCP: Locally weighted linear regression based approach for UCP estimation. *Appl. Intell.* **2023**, *53*, 13488–13505. [CrossRef]
31. Silhavy, R.; Bures, M.; Alipio, M.; Silhavy, P. Methodology, More Accurate Cost Estimation for Internet of Things Projects by Adaptation of Use Case Points. *IEEE Internet Things J.* **2023**, *10*, 19312–19327. [CrossRef]
32. Abrahao, S.; Insfran, E. A Metamodeling Approach to Estimate Software Size from Requirements Specifications. In Proceedings of the 34th Euromicro Conference Software Engineering and Advanced Applications, Parma, Italy, 3–5 September 2008; IEEE: Piscataway, NJ, USA, 2008. [CrossRef]
33. Sahoo, P.; Mohanty, J.R. Model-based Test Effort Estimator—A Case Study. In Proceedings of the 19th OITS International Conference on Information Technology (OCIT), Bhubaneswar, India, 16–18 December 2021; IEEE: Piscataway, NJ, USA, 2021. [CrossRef]
34. Alsaadi, B.; Saeedi, K. Data-driven effort estimation techniques of agile user stories: A systematic literature review. *Artif. Intell. Rev.* **2022**, *55*, 5485–5516. [CrossRef]
35. Usman, M.; Mendes, E.; Börstler, J. Effort estimation in agile software development: A survey on the state of the practice. In Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, Nanjing, China, 27–29 April 2015; ACM: New York, NY, USA, 2015. [CrossRef]

36. Sharm, A.; Kushwaha, D.S. Estimation of Software Development Effort from Requirements Based Complexity. *Procedia Technol.* **2021**, *4*, 716–722. [CrossRef]
37. Cohn, M. *Agile Estimating and Planning*; Pearson Education: London, UK, 2005.