Analyzing Software Architecture Documentation Models According to Agile Characteristics

Leonardo Barreto and Tayana Conte
Institute of Computing, Federal University of Amazonas, Manaus, Brazil

Keywords: Software Architecture, Agile Architecture, Architecture Documentation.

Abstract: **Background:** Software companies that use agile practices and methods usually postpone architecture design activities in favor of accelerated development and idea validation, especially in uncertain and dynamic contexts. However, this attitude leads to the accumulation of different types of technical debt, including architectural and documentation debt. As the company evolves, the architecture created during development becomes complex and hard to maintain, affecting the company's performance, the product's quality, and the knowledge transfer. **Aim:** Support the software architecture planning and documentation by verifying the feasibility of software architecture description approaches in the context of agile development. **Method:** We evaluated six approaches using the DESMET Feature Analysis method, with features related to implementation cost, flexibility, adaptation to dynamic requirements, usefulness, description consistency, decision analysis, and system modularity. **Results:** Two approaches had the best scores, with a minor percentage difference between them. These results are due to the low implementation cost of the two approaches, the factor that most influenced the score. **Conclusions:** The results provide evidence about the feasibility of applying the studied approaches, in agile contexts, besides reducing the number of possible alternatives for conducting experimental studies in this context.

1 INTRODUCTION

The software development process has changed over the years. Today, new software companies compete for market share by value delivery to customers through the launch of innovative high-tech products, using agile methods and practices (Kuhrmann et al., 2021). Most of these companies launch their solution idea to the market early to collect feedback from potential users. In this context, product ideas and features change constantly, as well as the requirements it must meet (Klotins et al., 2019b).

These companies spend resources and time on fast prototype development and early product versions. Quality-related aspects, such as architecture planning and automated tests, have low priority due to the dynamic and uncertain context those companies are in (Giardino et al., 2016). They also do not consider recording knowledge about the product so valuable due to constant changes altering or discarding the recorded knowledge leading to resources and time waste (Giardino et al., 2016). Furthermore, these companies consider documentation waste when it does not add value to the end product and measure the productivity as the amount of working software, turning documentation into a counter-productive task (Theunissen et al., 2022).

However, when the company grows and scales its product, it seeks to attract more investment, hire more employees, and define more organized development processes (Klotins et al., 2019b). The architecture developed was not optimally planned, and its knowledge, previously restricted to the company's founders, needs to be maintained by the new team members (Giardino et al., 2016). Additionally, the high amount of tacit knowledge makes it difficult to make decisions related to software architecture (Dasanayake et al., 2015), and it can create discrepancies between what the development team thinks about the system and how it works (Klotins et al., 2018).

In this sense, this research aims to verify, through a Feature Analysis (Kitchenham et al., 1996), the feasibility of applying software architecture documentation approaches in agile contexts. We evaluated the approaches according to relevant characteristics of software companies working in dynamic contexts, using agile practices or methods,

Barreto, L. and Conte, T.
Analyzing Software Architecture Documentation Models According to Agile Characteristics.
DOI: 10.5220/0011852300003467
In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023) - Volume 2*, pages 75-85
ISBN: 978-989-758-648-4; ISSN: 2184-4992

like software startups. As a contribution, this paper provides evidence about the evaluated approaches and narrows down the alternatives for software architecture documentation in agile contexts.

We divided the paper as follows: Section 2 presents the relevant theoretical background for this study; Section 3 describes the methodology; Section 4 presents the results; Sections 5 and 6 contain the discussion about the results and the limitations of the study, respectively, and we describe the conclusions in Section 7.

2 BACKGROUND

2.1 Agile Software Development

The Agile Manifesto (Beck et al., 2001) guides systems development to satisfy customers with rapid and continuous value delivery. Dynamic requirements are accepted and incorporated into software development, and the progress metric is the amount of software working. Some agile practices are iterative and incremental development, with new versions at short intervals and refactoring code from previous versions (Yang et al., 2016). Additionally, some methods aggregate these practices and help in the organization of the development process, such as Scrum, with sprints, to build incremental releases into one or two-week intervals, daily meetings for progress monitoring (Schwaber, 1997) and XP, through pair programming, continuous integration and refactoring and some other practices (Beck, 1999).

We can find the agile context in many companies, such as software startups (Klotins et al., 2021), due to the pressure of time-to-market, the need to acquire customers, and the uncertainty about the product's functionality and target audience. Pantiuchina et al. (2017) also state that software startups apply agile practices aimed at the speed of product delivery, such as frequent releases and short-term planning. However, quality-related practices, such as refactoring and automated testing, are neglected.

2.2 Agile Software Architecture and Documentation

ISO 42010:2022 describes software architecture as "the set of fundamental concepts or properties of a system, composed of its elements, relationships, and principles of design and evolution" (International Organization for Standardization, 2022).

Agile software architecture satisfies this definition and responds better to changing requirements, external factors, and uncertain contexts because it is easier to modify in an iterative, incremental manner (Waterman et al., 2015). Some strategies for creating agile architectures are to design just what is essential for the iteration, delay decision-making, accept an emergent architecture, and only plan the next iteration (Waterman et al., 2015).

Additionally, finding the balance between prior architecture planning and emergent architecture is a critical problem because professionals should make important decisions early, allowing the product to evolve and adapt (Wohlrab et al., 2019).

In this process, the system information and architecture decisions are available. However, companies do not value architecture documentation practices because, in their vision, they do not add value to the end product (Theunissen et al., 2022). The short-term focus also forces the team to focus on developing and maintaining the system (Theunissen et al., 2022). Professionals use informal communication and source code descriptions, hindering knowledge transfer, making it difficult to understand the system, especially for new members (Theunissen et al., 2022), and to make decisions about the architecture (Dasanayake et al., 2015).

Wohlrab et al. (2019) seek to solve the problems cited above by improving the consistency and usefulness of architecture descriptions with the following guidelines:

- Establish purposes and audiences for the descriptions, delimiting the levels of abstraction and the elements present in it.
- Separate the current architecture and future versions to facilitate the transition and the understanding of changes' impact.
- Document the minimum of elements of the future architecture, with only the most relevant that affect different parts of the company.
- Team evaluates architectural decisions before implementation.
- Integrate architects into the different teams to collect feedback, identify inconsistencies, and capture emerging aspects of the architecture as the system evolves.

3 METHODOLOGY

We performed a DESMET Feature Analysis (FA) to verify the feasibility of software architecture

documentation approaches in agile contexts. DESMET is a methodology for evaluating software engineering methods and tools proposed by Kitchenham et al. (1996). Examples of using DESMET Feature Analysis to evaluate techniques are the work of Parizi et al. (2020), in which the authors compared the proposed tool's features with other related tools using DESMET, and Marshall et al. (2014), where the authors compared, using DESMET, four tools that support the Systematic Review process to evaluate them. We chose this technique due to the restriction of time and resources for experimentation or questionnaire application on the six approaches in one company, as recommended by Kitchenham et al. (1996). The DESMET Feature Analysis requires a previously defined set of relevant features for a target audience. The steps for its realization involve candidates selection, features, subfeatures, weights and importance levels definition for each Feature Set, and candidates scoring using judgment scales.

3.1 Selected Candidates for Evaluation

We selected the approaches after an exploratory review in TOSEM, JSS, TSE, IST, and TOIS journals, searching for articles that present software architecture description approaches. Among the articles found, Hofmeister et al. (2007) present a framework for building architecture description models derived from five description approaches: 4+1, Siemens' 4 Views, Attribute Driven Design (ADD), BAPO/CAFCR, and Architecture Separation of Concerns (ASC). The first four are present in the Feature Analysis, and we did not include ASC due to the absence of the original article, and it is not possible to evaluate it.

Yang et al. (2016) present a mapping between agile methods and software architecture. One of the articles selected by the authors describes an approach that met the inclusion criteria, the C3A Agile Architecture, which we also evaluated. Finally, we performed a grey literature search to find approaches not described in articles published in the journals explored earlier. In this way, we found the C4 Model (Brown, 2022) approach and used it for evaluation. We listed the evaluated approaches below and further described them in Section 4.

Siemens' 4 Views. The approach proposed by Soni et al. (1995) has four main views: conceptual, module, execution, and code, describing elements, behaviors, and technologies used in the system.

Kruchten's 4+1. The 4+1 approach contains five views of a system's architecture, each related to different aspects of the system, like its structure or behavior. The views present are logical, process, development, physical, and scenario (Kruchten, 1995).

BAPO/CAFCR. The BAPO/CAFCR approach has five main views about different aspects of the system: business, application, functional, conceptual, and realization. It also has variations for the views, which role is to deal with usage scenarios (America et al., 2003).

Attribute Driven Design. The Attribute Driven Design approach has several recursive stages, done iteratively. The model focus on the quality attributes relevant to the stakeholders and the architecture. The views generated are the module, connector-component, and allocation views. (Wojcik et al., 2006).

C3A Agile Architecture. This approach has two mandatory levels of abstraction for the architecture and its separation into a reference version, which deals with major versions of the system, and an implementation version, which deals with future iterations (Hadar and Silberman, 2008).

C4 Model. The C4 approach has four main views, consisting only of boxes and lines that can represent any element. The views are system context, containers, components, and code. Besides these, the author presents other complementary views, such as the dynamic view and the development view (Brown, 2022).

3.2 Features, Weights, and Importance Levels Definition

In DESMET, it is necessary to define each set of features (Feature Set – FS) and their importance weight (IW). We defined and revised these weights iteratively, seeking adequacy to the context under study. In this study, the weights describe the relevance of the FS for agile contexts and based on (Marshall et al., 2014), the sum of the FS weights was equal to 1.0. Table 1 presents the FS with the description and weights associated with each.

We derived the features and importance weights from characteristics described in the literature about agile software development (Paternoster et al., 2014; Giardino et al., 2016; Klotins et al., 2019a;

Theunissen et al., 2022), agile software architecture (Waterman et al., 2015) and consistency and usefulness of architecture descriptions (Wohlrab et al., 2019).

The FS related to documentation (FS1 – $IW_{FS_1} = 0.15$) is intended to evaluate the support for recording knowledge about the software architecture provided by the candidate approaches. This set was also constructed to verify that the evaluated approach meets the criteria for a useful and consistent description (Wohlrab et al., 2019), in addition to the guidelines for building an agile architecture (Waterman et al., 2015): minimizing the number of planned elements in the architecture (FS1-02), the separation of audiences and purposes of the architecture views (FS1-03), and the differentiation between current and future instances of the architecture (FS1-04).

FS2 ($IW_{FS_2} = 0.05$) comprises the ability to analyze architectural decisions through the candidate approach, which includes recording the decisions made in artifacts and the reasoning performed during the decision-making. This FS was chosen to meet a criterion proposed by Wohlrab et al. (2019), to create useful and consistent descriptions. Additionally, agile companies postpone architectural decision-making as long as possible, due to the short-term focus, changing context, new objectives and new team members (Theunissen et al., 2022). However, the required and created knowledge for those goals disappears over the following iterations. Thus, facilitating analysis and decision-making in the early stages can contribute to the product evolution, reducing the debt that will accumulate.

FS3 ($IW_{FS_3} = 0.10$) is intended to evaluate candidates on their ability to design modular systems. This Feature Set was defined because modular construction of systems is a characteristic found in agile contexts, by allowing the removal or alteration of functionality, in a context of uncertain needs (Paternoster et al., 2014).

The fourth Feature Set (FS4 – $IW_{FS_4} = 0.35$) evaluates flexibility and adaptability to changing requirements, a characteristic very present in agile contexts (Giardino et al., 2016).

Finally, FS5 (FS05 – $IW_{FS_5} = 0.35$) evaluates the cost of applying the approaches, measured by the number of mandatory tasks to design the architecture. We defined this Feature Set because time is one of the most relevant resources for agile companies (Giardino et al., 2016). Furthermore, because working software is more important than documentation for these companies, this task can not take too much time and resources (Theunissen et al., 2022).

Development time and adaptation to dynamic requirements are of great importance for agile software companies (Giardino et al., 2016; Klotins et al., 2019a,b). Therefore, the FS related to them (FS4 and FS5) has an importance weight equal to 0.35, higher than the weight of the other FS. Since modular development (FS3) is a more encountered aspect than architectural decision-making (FS2), the weight of FS3 (0.10) is greater than that of FS2 (0.05). The weight of FS1 (0.15) is higher because it has more important subfeatures, such as the presence of few elements in the initial planning (FS1-02), ideal in the context of dynamic requirements.

We also need to define the **Importance Level (IL)** of each subfeature, inside the Feature Sets. It contemplates the relevance of the presence of a subfeature in the evaluated tool, for the target audience. The ILs have a multiplier value, used in the calculation of each candidate's score. The levels are 4-Mandatory, 3-Highly Desirable, 2-Desirable, and 1-Interesting. For example, we defined the subfeature FS4-01 ("The approach must be flexible and adaptable to changing requirements and architectural decisions.") as mandatory, then it has $IL_{FS4-01} = 4$.

3.3 Candidates Evaluation

To evaluate the candidate approaches (Subfeature judgment – SFJ_x) according to the defined features, it is necessary to define judgment scales. In this study, we used two scales: for FS1, FS2, FS3 and FS4, the scale is:

- **Yes - score 1**, if the subfeature is completely met by the approach.
- **Partially - score 0.5**, if the subfeature is not completely present or is implemented differently by the candidate.
- **No - score 0**, if the subfeature is not met.

For FS5, we assigned the score comparatively among the candidates, with scores from 1 to 6. We ordered the approaches in ascending order according to the number of mandatory tasks. We scored 6 for first place, 5 for the second, and so on. In the case of a tie, the candidates had the same evaluation. In this way, the approach with the fewest tasks would get the best score. When the approach does not present the tasks, we counted the number of mandatory views, presented by the authors of the candidate approach.

The Feature Set's maximum score (MS_{FS}) is the sum of the maximum values of its subfeatures. For example, FS1 has four subfeatures. FS1-01 was rated as Desirable ($IL_{FS1_{01}} = 2$). FS01-02 was rated Highly

Table 1: Feature Sets and subfeatures evaluated in the FA.

ID	Feature Set	SF Description	Importance Level (IL)	Importance Weight (IW)
FS1	Documentation	The approach should support the registration of knowledge about the system architecture. The approach should allow the documentation of a small amount of elements in the initial planning. The approach should have different descriptions of the architecture for different audiences and purposes. The approach must be able to separate current and future architectures.	Desirable (2) Highly Desirable (3) Interesting (1) Interesting (1)	0.15
FS2	Decision Rationale	The approach should facilitate the analysis of architectural decisions.	Interesting (1)	0.05
FS3	System	The approach must allow modularity of the system architecture.	Desirable (2)	0.10
FS4	Requirements	The approach must be flexible and adaptable to changing requirements and architectural decisions.	Mandatory (4)	0.35
FS5	Cost	The approach should allow the architecture to be built with the least amount of mandatory tasks.	Mandatory (4)	0.35

Desirable ($IL_{FS1_{02}} = 3$). The next two subfeatures were rated as Interesting ($IL_{FS1_{03}} = IL_{FS1_{04}} = 1$). In this case, it will be $MS_{FS1} = ((2 \times 1) + (3 \times 1) + (1 \times 1) + (1 \times 1)) = 7$.

The Feature Set x score ($\%FS_x^{Score}$) and the overall score (OS) of the applicants are calculated using the equations 1 and 2.

$$\%Score_{FS_x} = \frac{\sum(IL_{FS_x} \times SFJ_x)}{PM_{FS_x}} \quad (1)$$

$$OS = \sum(\%Score_{FS_x} \times IW_{FS_x}) \quad (2)$$

The first author documented a system using all approaches, to understand their application. Then the first author evaluated each approach, and all the authors discussed the scores.

4 RESULTS

In this section, we describe and justify the results of the Feature Analysis, using the information present in the seminal papers for each candidate approach. They can also be found in the tables at the end of each approach.

Siemens' 4 Views. Siemens' 4 Views approach (Table 2) supports architecture documentation ($Score_{FS1-01} = 1$) but does not discuss how many elements should be present in the initial architecture planning and design ($Score_{FS1-02} = 0$) (Soni et al., 1995). According to the authors, different teams (software architects, developers, operational staff, and others) use the four views because they address the results of decisions influenced by organizational and technical factors ($Score_{FS1-03} = 1$). Finally, the approach does not provide any separation between current and future architecture ($Score_{FS1-04} = 0$), it does not discuss how to document architectural decisions ($Score_{FS2-01} = 0$) or how to analyze a decision-making process.

This approach allows the construction of a modular architecture through functional decomposition and layering. The functional decomposition captures the system's distribution among modules, subsystems, and abstract units and the relationships between components through interfaces that can be exported and imported. The layered layout reflects the design decisions based on the import and export relationships, the constraints created, and the reduction and isolation of internal and external dependencies ($Score_{FS3-01} = 1$).

The authors state that the execution view tends to be modified the most among the others due to the need to support software and hardware evolution ($Score_{FS4-01} = 1$). The approach does not describe tasks to create the architecture descriptions. In this case, we counted the proposed mandatory views (four). On the scale adopted for FS5, we scored the approach ($Score_{FS5-01}$) as described below.

$$Score_{FS5-01} = 5 \therefore \%Score_{FS5} = \frac{5*4}{24} = \frac{20}{24} = 83.33\%$$

Table 2: FA results for the Siemens 4 Views approach.

			S4V				
Feature Set	FS Importance Weight	Subfeature ID	Importance Weight (IW)	Max. Score (MS)	Subfeature Judgement (SFJ_x)	Feature Set Score ($Score_{FS_x}$)	$\%Score_{FS_x}$
FS1	0.15	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 0 1 0	3.00	42.86%
FS2	0.05	FS2-01	1	1	0	0.00	0.00%
FS3	0.10	FS3-01	2	2	1	2.00	100.00%
FS4	0.35	FS4-01	4	4	1	4.00	100.00%
FS5	0.35	FS5-01	4	24	5	20.00	83.33%
					Overall Score (OS)		**80.60%**

Kruchten's 4+1. The 4+1 approach (Table 3) supports the documentation of the system architecture ($Score_{FS1-01} = 1$) based on scenarios that cyclically generate the views for small sets ($Score_{FS1-02} = 1$). Then the artifacts are validated by *stakeholders*, and the cycle repeats, with the scenarios not yet addressed (Kruchten, 1995). Each view has different purposes and audiences ($Score_{FS1-03} = 1$): system engineers use the Physical and Process views. End-users, customers, and data experts use the Logic view. Project managers and the implementation team see the system through the Development view.

As for the separation of current and future architecture instances, the author does not describe

any approach to this ($Score_{FS1-04} = 0$) (Kruchten, 1995).

The scenario view combines the other views and assists in the design validation for a given scenario. The design guidelines document stores the relevant decisions made, the architectural objectives, the scenarios, and the quality attributes, among other aspects of the system ($Score_{FS2-01} = 1$) (Kruchten, 1995).

The 4+1 approach allows the description of modular architectures through the development view, which contains the modular organization of the system ($Score_{FS3-01} = 1$). This view organizes subsystems into a hierarchy of layers, each providing a well-defined interface for communication with higher layers.

The authors guide the approach's usage iteratively because, after the first views, professionals know little to validate the architecture. At each iteration, it will be possible to insert or remove elements of the architecture and to accommodate changes that arise for whatever reason (Kruchten, 1995) ($Score_{FS4-01} = 1$).

Finally, the approach captures the most important functionalities in scenarios. In this sense, the approach describes four steps to build the architecture:

1. Define from a small set of scenarios based on their risks and importance.
2. Create abstractions that address the scenarios (classes, subsystems, and others).
3. Lay out the architectural elements in the four main views.
4. Implement, test, and validate the system, detecting flaws to correct.

On the scale adopted for FS5, we scored ($Score_{FS5-01}$) as described below.

$$Score_{FS5-01} = 5 \therefore \%Score_{FS5} = \frac{5*4}{24} = \frac{20}{24} = 83.33\%$$

Table 3: FA results for the 4+1 approach.

4+1							
Feature Set	**FS Importance Weight**	**Subfeature ID**	**Importance Weight (IW)**	**Max. Score (MS)**	**Subfeature Judgement (SFJ_x)**	**Feature Set Score ($Score_{FS_x}$)**	**$\%Score_{FS_x}$**
FS1	0.15	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 1 1 0	6.00	85.71%
FS2	0.05	FS2-01	1	1	1	1.00	100.00%
FS3	0.10	FS3-01	2	2	1	2.00	100.00%
FS4	0.35	FS4-01	4	4	1	4.00	100.00%
FS5	0.35	FS5-01	4	24	5	20.00	83.33%
					Overall Score (OS)		**92.02%**

BAPO/CAFCR. The BAPO/CAFCR approach (Table 4) (America et al., 2003) allows the architecture construction ($Score_{FS1-01} = 1$) through five views, using only the main scenarios to build the initial planning ($Score_{FS1-02} = 1$). The business view addresses the proposed value to customers, their motivations, and their needs. The application view describes the system's usage scenarios through activity flows, quality requirements, and domain models. The functional view describes the system's properties using, for example, a list of important system *features* or a matrix that maps them to the cost of implementing them. The conceptual view documents the essential concepts of the system to show the parts of the system and how they cooperate, as well as the design patterns and principles that govern development. Finally, the realization view contains the technologies used to implement the designed system aimed at the development team ($Score_{FS1-03} = 1$).

The approach does not distinguish between current and future architectures ($Score_{FS1-04} = 0$). The five main views and their variations help the architectural decision analysis at any point in the architecture design ($Score_{FS2-01} = 1$). The conceptual view describes modular systems, with the elements and their relationships presented in the system decomposition ($Score_{FS3-01} = 1$).

The variations presented deal with different system usage scenarios, contain emerging requirements, and represent future (or new) impacts on the architecture. Changes in the architecture should be treated as new scenarios and modeled as existing scenarios ($Score_{FS4-01} = 1$). The authors do not define tasks to build the architecture (FS5-01). In this case, we counted the mandatory views presented. On this approach, there are five views (business, application, functional, conceptual, and realization). On the scale adopted to evaluate SF5-01, the CAFCR approach had the $Score_{FS5-01}$ calculated as described below.

$$Score_{FS5-01} = 4 \therefore \%Score_{FS5} = \frac{4*4}{24} = \frac{16}{24} = 66.67\%$$

Table 4: FA results for the BAPO/CAFCR approach.

CAFCR							
Feature Set	**FS Importance Weight**	**Subfeature ID**	**Importance Weight (IW)**	**Max. Score (MS)**	**Subfeature Judgement (SFJ_x)**	**Feature Set Score ($Score_{FS_x}$)**	**$\%Score_{FS_x}$**
FS1	0.15	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 1 1 0	6.00	85.71%
FS2	0.05	FS2-01	1	1	1	1.00	100.00%
FS3	0.10	FS3-01	2	2	1	2.00	100.00%
FS4	0.35	FS4-01	4	4	1	4.00	100.00%
FS5	0.35	FS5-01	4	24	4	16.00	66.67%
					Overall Score (OS)		**86.19%**

Attribute Driven Design. The Attribute Driven Design (ADD) approach (Table 5) helps in documenting the system architecture ($Score_{FS1-01} = 1$), focusing on quality attributes. $Score_{FS1-02} = 0.5$ because the stakeholders must prioritize all requirements before the architecture design begins. However, the authors also state that this is hardly possible and recommend that the architect work with the requirements he has at hand (Wojcik et al., 2006). During the architecture design, users create three views with different purposes but without defined audiences. Thus, SF1-03 is partially satisfied and thus $Score_{FS1-03} = 0.5$. The module view is for documenting the static system's properties. The component-connector view is concerned with the system's execution behavior. The allocation view maps the software elements to the hardware components. The approach does not have a separation of current and future architecture ($Score_{FS1-04} = 0$).

The approach has a step-by-step approach to building the software architecture, and from the fourth stage, the authors state that design decisions start ($Score_{FS2-01} = 1$). They involve the choice of system design concepts, elements, allocated resources, internal and external dependencies, and the validation of quality attributes. ADD allows the construction of modular systems ($Score_{FS3-01} = 1$) in the module view, which contains the system's elements. Users of ADD should address flexibility and adaptation to design changes at the end of each task. They must analyze to verify if the system's architecture is according to the planned and, if necessary, change and refine the generated architecture ($Score_{FS4-01} = 1$).

About the cost of the ADD approach, it has eight stages in each iteration. These are

1. The stakeholders prioritize all the requirements.
2. Confirm that there is enough information to build the architecture and rank all requirements in order of importance for the stakeholders.
3. Choose and decompose a system element according to the need.
4. Identify the requirements that drive the development and rank them according to their impact on the architecture and importance to the stakeholders.
5. Choose a design concept that satisfies the architectural drivers and evaluate and resolve inconsistencies in the design concept.
6. Instantiate architectural elements and allocate responsibilities in a module, component-connector, and allocation views.
7. Exercise the functional requirements instantiated in step VI, noting the inputs and outputs of the elements and documenting the interface for each element.
8. Verify and refine the requirements.

Because it was the approach with the most tasks, on the scale adopted for the SF5-01, we scored it with the lowest value.

$$Score_{FS5-01} = 2 \therefore \%Score_{FS5} = \frac{2*4}{24} = \frac{8}{24} = 33.33\%$$

Table 5: FA results for the ADD approach.

ADD							
Feature Set	**FS Importance Weight**	**Subfeature ID**	**Importance Weight (IW)**	**Max. Score (MS)**	**Subfeature Judgement** (SFJ_x)	**Feature Set Score** ($Score_{FS_x}$)	$\%Score_{FS_x}$
FS1	0.15	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 0.5 0.5 0	4.00	57.14%
FS2	0.05	FS2-01	1	1	1	1.00	100.00%
FS3	0.10	FS3-01	2	2	1	2.00	100.00%
FS4	0.35	FS4-01	4	4	1	4.00	100.00%
FS5	0.35	FS5-01	4	24	2	8.00	33.33%
					Overall Score (OS)		**70.24%**

C3A Agile Architecture. The C3A (Table 6) serves to build the architecture ($Score_{FS1-01} = 1$) of the system in agile contexts, with a minimum amount of elements ($Score_{FS1-02} = 1$). There are two main views: the reference architecture (RA), which shows a system's overview, with the main modules and their components, and the implementation architecture (IA), describing the changes in the architecture for new and minor versions, as well as more details on how to develop the system. The reference architecture contains the architecture layers, with level 0 and level 1 elements, the visionaries elements, and, optionally, deeper levels of elements. Business analysts use the business layer to record the system's integration with external systems. Software architects and project managers describe the interfaces between systems and users in the functional layer. Software architects use the system architecture layer to comprise the system's internal parts and implement the functionality described in the previous layer ($Score_{FS1-03} = 1$). The C3A approach has a clear separation between the current architecture ($Score_{FS1-04} = 1$), represented by the RA, and future versions, projected in the IA.

The approach's users must create a contract for each level 0 and level 1 element to register the architectural decisions, with relevant information to help in the analysis of the decisions: Name, Owners, Responsibilities, Dependencies, Implementation, API, Data structure, Technologies, among others ($Score_{FS2-01} = 1$). The Reference and

Implementation architectures enable the construction of modular systems containing the level 0 and 1 elements, called modules and components, respectively. The system is described in its main modules and decomposed into smaller components, which further explain the system's operation ($Score_{FS3-01} = 1$).

The approach has visionary or strategic components representing new features or changes, which may or may not be added to the system. Their presence helps align the position and value of the new *features* with the overall system's architecture and the likely impact caused. These components are only added to the RA when they are deemed mature by the team ($Score_{FS4-01} = 1$).

The step-by-step described to build the architecture has six steps (Hadar and Silberman, 2008):

1. Collect architectural documents, technical publications, manuals, and tacit knowledge, and capture the first architecture in a reference architecture, with the definition of level 0 components.
2. Validate the design created so far, detailing the status of the modules of future reference architectures while maintaining the current one to prevent it from being impacted by uncertain components.
3. For each level 0 module, define the level 1 components and implement the incremental releases of the RAs.
4. Map or provide gap analysis between the upcoming releases and the current architecture, and adjust the IA without impacting the RA. If not possible, mitigate the impact on the RA.
5. Modify minor release schedules to fit the current architecture, then execute and implement the IA.
6. Estimate the effects of the IA on the next release of the RA, propagating the gaps found in the last IA. If necessary, return to step I. Otherwise, return to step IV.

In the ranking adopted to score each candidate's FS5-01, C3A had the $Score_{FS5-01}$ calculated as described below.

$$Score_{FS5-01} = 3 \therefore \%Score_{FS5} = \frac{3*4}{24} = \frac{12}{24} = 50.00\%$$

C4 Model. The C4 approach (Table 7) allows the documentation of the system architecture through four main views ($Score_{FS1-01} = 1$). With a small set of information, it is possible to build the context and container views, generating an initial summary system's design with the main actors and components ($Score_{FS1-02} = 1$).

Table 6: FA results for the C3A approach.

C3A							
Feature Set	**FS Importance Weight**	**Subfeature ID**	**Importance Weight (IW)**	**Max. Score (MS)**	**Subfeature Judgement** (SFJ_x)	**Feature Set Score** ($Score_{FS_x}$)	$\%Score_{FS_x}$
FS1	0.15	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 1 1 1	7.00	100.00%
FS2	0.05	FS2-01	1	1	1	1.00	100.00%
FS3	0.10	FS3-01	2	2	1	2.00	100.00%
FS4	0.35	FS4-01	4	4	1	4.00	100.00%
FS5	0.35	FS5-01	4	24	3	12.00	50.00%
					Overall Score (OS)		**82.50%**

The views have different audiences and purposes ($Score_{FS1-03} = 1$): the context view involves only the system in question, the external systems, and other actors that interact with it, used by external people or members of the development team; the container view contains the system's subsystems, main parts and the actors that interact with them, used by the development team, operational people and software architects; the component and code views are more technical and describe the elements that make up the containers, and descriptions in lines of code or UML diagrams with the system implementation, aimed at developers and software engineers.

The approach does not distinguish between current architecture and future architectures ($Score_{FS1-04} = 0$) and does not propose any method for analyzing architectural decisions ($Score_{FS2-01} = 0$).

C4 allows the construction of modular architectures with the container view ($Score_{FS3-01} = 1$), describing the modules (subsystems) of the system, their relationships to each other, and the actors. Similarly, users can break the containers into smaller modules called components. The author states that each diagram will change at different speeds ($Score_{FS4-01} = 1$), with the component view changing the most as the team adds or removes these elements.

There are no defined tasks to build the architecture. In this case, we counted the mandatory views. The approach has four views, but the author states that only the context and container views are necessary and will be used in most companies and systems Brown (2022). On the scale used to evaluate FS5-01, C4 had the $Score_{FS5-01}$ calculated as described below.

$$Score_{FS5-01} = 6 \therefore \%Score_{FS5} = \frac{6*4}{24} = \frac{24}{24} = 100\%$$

Table 7: FA results for the C4 approach.

C4							
Feature Set	**FS Importance Weight**	**Subfeature ID**	**Importance Weight (IW)**	**Max. Score (MS)**	**Subfeature Judgement** (SFJ_x)	**Feature Set Score** ($Score_{FS_x}$)	$\%Score_{FS_x}$
FS1	0.15	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 1 1 0	6.00	85.71%
FS2	0.05	FS2-01	1	1	0	0.00	0.00%
FS3	0.10	FS3-01	2	2	1	2.00	100.00%
FS4	0.35	FS4-01	4	4	1	4.00	100.00%
FS5	0.35	FS5-01	4	24	6	24.00	100.00%
					Overall Score (OS)		**92.86%**

Table 8: Ranking of the evaluated approaches.

#	1st	2nd	3rd	4th	5th	6th
Approach	C4	4+1	CAFCR	C3A	S4V	ADD
Overall Score (OS)	92,86%	92,02%	86,19%	82,50%	80,60%	70,24%

5 DISCUSSIONS

Giardino et al. (2016) report that accelerated development causes startups and small companies that adopt agile practices to postpone planning and quality activities. Theunissen et al. (2022) also state that the short-term focus is an obstacle to documenting knowledge in agile contexts because professionals can code or maintain the system without registering decisions, rationale, and other aspects.

In this sense, it is possible to consider an approach that allows architecture construction and documentation in a little information context, with dynamic and uncertain requirements, under time-to-market pressure as a good solution for agile software companies.

As seen in Table 8, the best-rated approaches that perhaps fit the above definition are the C4 ($OS = 92.86\%$) and 4+1 ($OS = 92.02\%$) approaches, and the percentage difference between them is insignificant. Thus, the order between them can change through further analysis.

Generally, most approaches allow the description of the architecture with few elements. However, ADD is the only approach that requires the elicitation of all system requirements before the architecture design. Still, the authors also state that this context is almost unachievable (Wojcik et al., 2006).

Regarding purposes and audiences separation for the designed visions, most approaches explain which stakeholders benefit from each view and the system elements in each. We highlight CAFCR, C4, and 4+1 because they describe both abstract and more specific visions.

The Commercial view (America et al., 2003) and the Context view (Brown, 2022) aim at a general audience, with few internal and technical elements, describing the motivations and needs of the users, as well as the main system's elements and parts that interact with it. In the 4+1 (Kruchten, 1995) approach, the views detail different aspects of the system that, in the scenario view, are integrated to describe and validate how the system will meet different system usage scenarios.

The ADD (America et al., 2003), C3A Agile Architecture (Hadar and Silberman, 2008), and 4+1 (Kruchten, 1995) approaches stand out regarding architectural decision analysis. The former (America et al., 2003) proposes to validate what has been designed, against system and stakeholder needs, in half of the tasks required to build the architecture. The C3A (Hadar and Silberman, 2008) suggests creating contracts for the system elements with relevant information from the modules and components. The 4+1 (Kruchten, 1995) also guides the creation of a document called Design Guidelines, containing the major decisions made.

All the evaluated approaches provide means of building modular architectures for systems. Contracts and diagrams describe the system's elements with generic boxes and arrows or in UML language. We highlight Siemens' 4 Views (Soni et al., 1995) and 4+1 (Kruchten, 1995) approaches, which in 1995 already had the architecture description of system modules. The first approach allows this description through functional decomposition and layered organization, while the second approach organizes the modules in the logical view of the architecture.

Regarding flexibility and the ability to adapt the architecture to possible changes, all approaches proved to be capable of being modified. The approaches C3A (Hadar and Silberman, 2008) and CAFCR (America et al., 2003) have similar ways of changing the system architecture. The first has an implementation architecture containing visionary components, whose function is to describe new functionalities or changes, evaluated as to the impact on the architecture until they reach maturity for inclusion in the reference architecture. The CAFCR proposes the use scenarios to store the emerging requirements and represent future or new impacts on the architecture.

Regarding the implementation cost, we evaluated ADD and C3A as the most expensive. ADD has eight main steps (Wojcik et al., 2006) and C3A (Hadar and Silberman, 2008) have six tasks involving, for example, collecting information about the system and its functionalities, defining the first versions of the architecture, detailing and combining the planned future modules with the current version of the system.

6 LIMITATIONS

The limitations of this study are the choice of the evaluated features, the weight definition, and each candidate's cost evaluation. The risk generated by these limitations is that the weights used may not reflect the true importance of the factors, and the results cannot be generalized. To minimize this risk, we extracted the features and weights used from the characteristics reported in the literature by the work performed with interviews and surveys in the context of agile development.

Additionally, we can relate the cost of implementing an approach to the complexity of each task. However, its measurement may differ depending on who performs the task. Therefore, we only used the number of required tasks to evaluate this aspect more objectively. Still, we could not measure the number of mandatory tasks (FS5) of all candidates due to the absence of this information in the original articles. We used only the number of mandatory views to circumvent this problem since they are the artifacts to be generated and the task to which the team will allocate their effort.

7 CONCLUSIONS

Although agile companies value working software over system documentation, it can help with product evolution and quality improvement in future releases. In this sense, we evaluated six software architecture documentation approaches in agile contexts through a DESMET Feature Analysis process, defining a set of features the approaches must possess according to the agile software development context described in the literature.

The C4 (Brown, 2022) and 4+1 (Kruchten, 1995) approaches obtained the highest scores and may be the best recommended for application in agile companies. These companies can use the C4 approach to register an initial software architecture, including relevant actors, subsystems, and components, evolving it in release cycle iterations. They can also use the 4+1 approach to document physical aspects, system behavior, use scenarios, or code organization.

In the future, we will use these two approaches in experimental studies with industry professionals to collect participants' perceptions about the approaches and to evaluate the results presented in this paper. We appreciate the reviewers' suggestions and added a study about the correlation between development tools and agile architecture to future work plans.

ACKNOWLEDGEMENTS

We thank the financial support granted by Research and Development (R&D) project 001/2020, signed with Universidade Federal do Amazonas and FAEPI, Brazil, which has funding from Samsung, using resources from the Informatics Law for the Western Amazon (Federal Law No. 8.387/1991) and its disclosure is following article 39 of Decree No. 10.521/2020. We also thank the support provided by Universidade Federal do Amazonas (UFAM), CAPES - Financing Code 001, CNPq process 314174/2020-6, FAPEAM process 062.00150/2020, São Paulo Research Foundation (FAPESP) process 2020/05191-2, and the USES research group.

REFERENCES

America, P., Rommes, E., and Obbink, H. (2003). Multi-view variation modeling for scenario analysis. In *International Workshop on Software Product-Family Engineering*, pages 44–65. Springer.

Beck, K. (1999). Embracing change with extreme programming. *Computer*, 32(10):70–77.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development.

Brown, S. (2022). *The C4 model for visualising software architecture*. Leanpub.

Dasanayake, S., Markkula, J., Aaramaa, S., and Oivo, M. (2015). Software architecture decision-making practices and challenges: an industrial case study. In *2015 24th Australasian Software Engineering Conference*, pages 88–97. IEEE.

Giardino, C., Paternoster, N., Unterkalmsteiner, M., Gorschek, T., and Abrahamsson, P. (2016). Software development in startup companies: The greenfield startup model. *IEEE Transactions on Software Engineering*, 42(6):585–604.

Hadar, E. and Silberman, G. M. (2008). Agile architecture methodology: long term strategy interleaved with short term tactics. In *Companion to the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications*, pages 641–652.

Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., et al. (2007). A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126.

International Organization for Standardization (2022). Software, systems and enterprise – architecture description. Standard Iso/iec/ieee 42010:2022,

International Organization for Standardization, Geneva, CH.

Kitchenham, B., Linkman, S., and Law, D. (1996). DESMET: A method for evaluating software engineering methods and tools. Technical report, Department of Computer Science, University of Keele, Keele, Staffordshire, ST5 5BG, U.K.

Klotins, E., Unterkalmsteiner, M., Chatzipetrou, P., Gorschek, T., Prikladnicki, R., et al. (2018). Exploration of technical debt in start-ups. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 75–84. IEEE.

Klotins, E., Unterkalmsteiner, M., Chatzipetrou, P., Gorschek, T., Prikladnicki, R., et al. (2019a). A progression model of software engineering goals, challenges, and practices in start-ups. *IEEE Transactions on Software Engineering*.

Klotins, E., Unterkalmsteiner, M., Chatzipetrou, P., Gorschek, T., Prikladnicki, R., et al. (2021). Use of agile practices in start-up companies. *e-Informatica Software Engineering Journal*, 15(1).

Klotins, E., Unterkalmsteiner, M., and Gorschek, T. (2019b). Software engineering in start-up companies: An analysis of 88 experience reports. *Empirical Software Engineering*, 24(1):68–102.

Kruchten, P. B. (1995). The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50.

Kuhrmann, M., Tell, P., Hebig, R., Klünder, J., Münch, J., Linssen, O., Pfahl, D., Felderer, M., Prause, C. R., MacDonell, S. G., et al. (2021). What makes agile software development agile? *IEEE Transactions on Software Engineering*, 48(9):3523–3539.

Marshall, C., Brereton, P., and Kitchenham, B. (2014). Tools to support systematic reviews in software engineering: a feature analysis. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, pages 1–10.

Pantiuchina, J., Mondini, M., Khanna, D., Wang, X., and Abrahamsson, P. (2017). Are software startups applying agile practices? the state of the practice from a large survey. In *International Conference on Agile Software Development*, pages 167–183. Springer, Cham.

Parizi, R., da Silva, M. M., Couto, I., Marczak, S., and Conte, T. (2020). A design thinking techniques recommendation tool: An initial and on-going proposal. In Viana, D. and Schots, M., editors, *19th Brazilian Symposium on Software Quality, SBQS 2020, São Luís, Brazil, December, 2020*, page 36. ACM.

Paternoster, N., Giardino, C., Unterkalmsteiner, M., Gorschek, T., and Abrahamsson, P. (2014). Software development in startup companies: A systematic mapping study. *Information and Software Technology*, 56(10):1200–1218.

Schwaber, K. (1997). Scrum development process. In *Business Object Design and Implementation*, pages 117–134. Springer.

Soni, D., Nord, R. L., and Hofmeister, C. (1995). Software architecture in industrial applications. In *1995 17th International Conference on Software Engineering*, pages 196–196. IEEE.

Theunissen, T., van Heesch, U., and Avgeriou, P. (2022). A mapping study on documentation in continuous software development. *Information and Software Technology*, 142:106733.

Waterman, M., Noble, J., and Allan, G. (2015). How much up-front? a grounded theory of agile architecture. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 347–357. Ieee.

Wohlrab, R., Eliasson, U., Pelliccione, P., and Heldal, R. (2019). Improving the consistency and usefulness of architecture descriptions: Guidelines for architects. In *2019 IEEE International Conference on Software Architecture (ICSA)*, pages 151–160. IEEE.

Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., et al. (2006). Attribute-driven design (ADD), version 2.0. Technical report, Software Engineering Institute, Carnegie-Mellon University.

Yang, C., Liang, P., and Avgeriou, P. (2016). A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, 111:157–184.