Adopting the C4 model for lightweight architecture modeling – An experience report ⋆

Robbert Jongeling[1][0000−0002−1863−3987], Niels Jørgen Strøm[2], Lars Peter Torp Nissen[2], Martin Kitchen[2], and Jan Carlson[1][0000−0002−8461−0230]

[1] Mälardalen University, Västerås, Sweden
{robbert.jongeling,jan.carlson}@mdu.se
[2] Grundfos, Denmark {njstroem,lnissen,mkitchen}@grundfos.com

Abstract. This paper reports on experiences of migrating to a new way of documenting software architecture for embedded software development at a company developing software-intensive systems. A lightweight modeling approach using the C4 model for software architecture was adopted, where the architecture diagrams were expressed in PlantUML and stored within Markdown files alongside the code. We describe the problem context and the new approach, and report on a questionnaire survey among the software engineers to elicit their experiences and opinions two years after the introduction of this way of working. The majority of the 48 respondents reported being successful in using the new way of working and see benefits in terms of the better organization of the documentation as stored alongside the code, a clearer process of documenting, better alignment with the code, and ease of use. Engineers report a manageable learning curve but foresee challenges in maintenance and communication of changes across teams working on embedded software, hardware and electronics.

Keywords: Software architecture modeling · C4 model · Adoption in practice

1 Introduction

Organizations rely on various ways to document their software architecture, and these ways are not uncommonly guided by convenience and legacy. When change to the architecture documentation practice is needed, it may be challenging to choose an appropriate method among the many available options. In this paper, we provide an experience report of such a change at a company, not claiming a definitive best solution to a problem, but contributing to the body of knowledge that may support decisions about documenting architecture in similar situations.

The context of this research is a long-running collaboration with the partner company, in which we have been working on ways to improve their software architecting practice. This paper reports on a change in the way of working implemented by the company two years ago, we describe the change and survey

⋆ This research was supported by Software Center www.software-center.se.

various stakeholders to map their experiences. The company develops cyber-physical systems and embedded software for these systems. Due to an increase in the required capabilities of these systems and a growth of variants of the products the company offers, the number of development teams in the company has also increased. Consequently, the previous ways of capturing the software architecture were deemed no longer sufficient. To address the shortcomings, the company has adopted a hierarchy of documenting the architecture following the C4 model for software architecture by Simon Brown [1]. This model consists of four layers: Context, Containers, Components, and Code. Entities at each layer are contained in entities in the above layers. The context level captures systems and people, the containers level shows the containers that constitute a software system of interest, and so on. In addition to adopting this hierarchy, the company has established new modeling guidelines for the various software engineering teams to create documentation in Markdown files, including PlantUML or "draw.io" diagrams, and store them alongside the code in the same repositories.

We report on experiences of this solution two years after its introduction. We gather feedback in discussions with the responsible architects and through a questionnaire among the software engineers. We find that the engineers perceive benefits in standardizing the way of working, in the structure provided by the C4 model, and by having the documentation alongside the source code. Reported challenges include a learning curve (although manageable), layout issues with PlantUML, and specific challenges encountered during the migration to the new way of working. Foreseen challenges for the future relate to maintenance.

The remainder of this paper is organized as follows. Section 2 describes the previous state of practice at the company and the need for adopting a new approach for documenting the architecture. Section 3 briefly mentions prior work exploring the same problem space as discussed in this paper. Section 4 describes the process of making the changes to the new way of working, and presents details of the chosen lightweight modeling to documenting the architecture, following the C4 model for architecture and PlantUML diagrams embedded in text files. Section 5 presents the results of an evaluative questionnaire answered by 48 software engineers in the company. Section 6 provides a discussion and reflection on the experiences reported on in this paper. Section 7 concludes.

2 Problem context

The company in which we performed our study develops several products containing embedded software. This software is developed by approximately 150 software engineers, divided over eight platform teams and nine product teams that focus on specific products. Additionally, there are teams working on mobile and cloud applications, but these are not the focus of this experience report, which is limited to the domain of embedded software development. We have worked closest together with one central architecture team, responsible (among other things) for establishing the way of working with software architecture documentation. In particular, we focus on the way in which the software architecture and design are documented. We briefly describe the previous way of working here and the new way of working in Section 4.

The prior way of documenting the architecture was done through a combination of various loosely coupled artefacts. To describe features, a feature model was expressed in Pure::Variants [2]. Textual requirements were captured in IBM Doors, and an implementation in C code. The architecture was documented in Office documents stored in SharePoint, disconnected from the code. Feature definitions were strictly linked to requirements, but no strict linking was present between feature definitions and their implementation and verification artefacts. This was handled manually.

This way of working had several challenges, as identified by the architects and software engineers. The architecture and design documentation was difficult to locate. Consequently, it was heavy to maintain these artefacts. Moreover, these artefacts were non-uniform, with potential big differences across teams. This was partly due to the informal nature of the architecture description and partly due to the design process, which did not explicitly include any modeling requirements as part of the documentation effort. The lack of a more guided way of working meant that changes to the implementation could be made without corresponding changes in the documentation, and this would potentially go unnoticed. Such inconsistencies then extended to other artefacts too, e.g., the feature model. These challenges impacted both the platform and the product teams. Therefore, the architecture team decided to implement changes in the way the architecture was documented across the teams working on embedded software development.

3 Related work

In this paper, we report on experiences of modeling the architecture in textual (PlantUML) models that are created in a regular text editor and not in a dedicated modeling tool. The models have underlying semantics and the way they are expressed enables some degree of automated support for, e.g., consistency checking. Diagrammatic representations of these PlantUML models are automatically rendered and included in the documentation. This situates the modeling practices between informal diagramming and canonical modeling [3]. The considerations that the company made when introducing the practices relates to the question of "how much modeling is enough" [4]. Some modeling is needed for having an accurate documentation of the system, but complete modeling of the entire software has not enough benefits to be worth the cost. Hence, this in-between solution was chosen.

Software architecting is a continuous activity [5]. A consequence is that the architecture is subject to change throughout development, due to, e.g., new incoming requirements. A common challenge in practice is then the maintenance effort of the architecture and ensuring its alignment with the source code [6]. These consistency problems are common and not new. Indeed, the established approach of inconsistency tolerance [7] is still relevant [8].

We mention three experience reports on architecting of software-intensive systems that are relevant to this paper. The first focuses on re-use of architecture decisions and finds that this is done ad-hoc [9]. This is relevant in our setting too because of the shared embedded software domain and platform-product setup. A second experience report is on using the C4 model in education [10], where the C4 model is also used in conjunction with UML modeling. Experiences from practice are more readily found in the gray literature than in research papers. The third experience report we mention states that using diagrams at all four levels is not necessary and it is better to pick and choose which ones are relevant in the particular setting [11].

4 From the previous to the new way of working

During 2021, the architecture team has identified the challenges discussed in Section 2. One of the challenges was how to deal in particular with consistency checking between the source code and the architecture, which at that time was captured only in informal diagrams. During their early work, the team concluded that there was a lack of documentation at the lower levels of abstraction. In particular, besides an overview of the whole system, there was a need to capture the design of individual components and their interfaces. Initial work in this direction revealed the need to delimit more clearly what a software component entails.

4.1 Characterizing a software component

The effort of clarifying what a software component is has resulted in the following description of its characteristics. A software component defines the level of reusability and is captured at the C4 components level. To ensure it can be used independently, a component must have high cohesion and low coupling. Moreover, components of the embedded software shall be built into an executable which is deployed on a single ECU. Executables cannot be distributed across multiple ECUs, and therefore neither can components.

Software components are a crucial element in the hierarchical definition of the systems under development. At the highest level, there is a family of products. Features are defined in a software product line context in a feature modeling tool (Pure::Variants). Each feature is implemented by a single software component. Software components are grouped using the same structure as the features. Naming conventions are in place to amplify alignment between the diagrams and the source code.

4.2 Goals for the new way of working

The following goals for the new way of working with documentation were formulated by the company:

1. To have one holistic approach to architecture and design documentation;

2. For the documentation to be easy to locate, easy to produce, and easy to maintain;
3. To obtain more uniform documentation across development teams;
4. To use standard diagrams.

The new way of documenting comprises a textual documentation of the architecture in Markdown files, which will be stored alongside the source code and placed under version control. The Markdown files contain PlantUML definitions of diagrams, following the modeling semantics of UML. Detailed system aspects and connection to other (software) domains are currently captured partly outside this framework. This is a challenge which is discussed in Section 5.

4.3 C4 model adoption

After establishing the definition of software component, the architects have also provided guidelines of what and how to model at each of the C4 levels. These guidelines for each level are included in brief below.

C4 – Context The intent of diagrams at this level is to identify the software systems that are built and delivered, in addition to who are using these systems. Moreover, they document how software systems interface with the existing environment. The context diagrams are described for each product instance. Stakeholders are product specialists, software engineers, and other product development engineers. The diagrams represent a static view of the context of the system and interactions with other software systems or people. In some cases, dynamic views are needed, for which scenario views are defined. These contain the same elements and are represented using UML sequence diagrams or communication diagrams.

C4 – Containers The intent of diagrams at this level is to identify software containers that are built and delivered. The diagrams shall show how software containers are deployed on hardware elements, and how software containers interact with other containers or external systems. At this level, one view is required: a deployment view, which is a static view of the software containers deployed on hardware elements. A container in this context represents a software program developed by the company. Containers can be considered executables or deployables, too. Versioned releases of containers are published internally.

The deployment view comprises containers and their interactions with external systems. The container diagrams are created for each product instance. Stakeholders are product specialists, software engineers, and other product development engineers. Also at this level, sometimes dynamic scenario views are defined, using UML sequence diagrams or communication diagrams.

C4 – Components Inside a container, we find components. One mandatory view that is created for each product instance shows the software components and their relations. This view is derived from a "150%" diagram (a single view including the union of the components used in the all product variants [12]), by excluding components that are not used in the product. Sometimes, new components can be added, to both this diagram and the "150%" diagram. The intent of diagrams at this level is to identify which software components are available and which are going to be included in the build of the instance. Moreover, the diagrams are used to illustrate how the software components interact.

Diagrams showing the component view are separated into two drawings, one for the basic layer and one for the application layer on top of the basic layer. Due to the stringent definition of software component, the component view for the application layer can be automatically generated for both the platform and product instances. Stakeholders are software engineers. For the basic layer, due to legacy and the importance of the graphical layout (visual placement of boxes) for conveying information about groupings and the relationships between components, component view diagrams are created using a drawing tool (draw.io). The drawing tool is preferred in this case over PlantUML, since the latter by design does not allow fine-grained control over layout of graphical elements.

Finally, dynamic views may be created that show scenarios using UML sequence, collaboration, communication, or activity diagrams.

C4 – Code The intent of diagrams at this level is to serve as a manual or user guide for users of the component, to be a reference manual for the public interface of the component, and to capture the internal design of the component to facilitate maintenance. Diagrams at the code level describe modules and classes, and how they are related. Component documentation must explain how a user of the component shall instantiate and configure it. It must be clear how to reuse the component, connect to other components it depends on, scale it, and activate its functionality.

Internal design documentation could be represented by state machines. For dynamic scenarios, sequence or communication diagrams can be used. Component documentation is created at platform level, written in PlantUML in Markdown files stored alongside code in the same folder. Stakeholders are software engineers working on development of the embedded software.

4.4 Process changes

In addition to adoption of the C4 model, there are some other changes to the way of working. For example, a gold-silver-bronze medal incentive system is introduced to motivate teams to create higher documentation coverage and to be able to monitor progress in the refactoring effort. The main purpose of these incentives if to increase alignment between feature model and code base.

5 Findings and insights

In this section, we present insights gathered through discussions with the architecture team and through a questionnaire sent out to software engineers within the company. The architecture team that was responsible for initiating the changes to the way of working, reports five perceived benefits and three open challenges from their perspective.

5.1 Expert opinions from the architecture team

Experienced benefits of the new way of working The first experienced benefit is a closer connection between the documentation and the code. The lightweight way of modeling, i.e., easy to produce and read but still having a defined semantics, allows for modeling to be a natural part of the design work. Moreover, the documentation is now co-located with the code, which makes it more accessible for software engineers to view. Producing documentation models is also made easier, since it is integrated in the daily work environment of software engineers, through tools they are already using (Visual Studio Code and BitBucket). These benefits are thus seen in readily available documentation and an easier process by which to produce and maintain it.

A second experienced benefit is in the ability to identify changes without heavy tooling. Most tooling that is needed was already established. The architecture is now expressed in a textual format and stored under version control, allowing easy identification of changes using text comparison. In addition, some new tooling is required for rendering the PlantUML diagrams, but this can be done within Visual Studio Code, that was already used, in combination with browser extensions for viewing the documents in BitBucket.

Benefits of following the C4 model for architecture are also reported. Modeling at the C4 *Components* level has allowed for the automatic generation of static views showing the included and excluded components for particular products. In addition, a dynamic view describes behavior belonging to the platform level, and these aspects are not expected to change in a product instance. Diagrams at the higher levels of the C4 model, specifically *Context* and *Containers* were found useful for communication with non-software stakeholders. These benefits are thus related to the choice to follow the C4 model for the various levels of abstraction at which the architecture is expressed.

Open challenges Despite the change in way of working, a persisting challenge is monitoring the consistency between various artefacts. While in the new way of working, the process of updating the documentation is closer to the software development, and the documentation is placed closer to the code, no automated means are in place to check the alignment between documentation and implementation. Thus the teams rely on manual means, e.g., code reviews to ensure that when code changes are made, the relevant documentation is also updated. This consistency is nevertheless important, especially when other teams rely on the models as an accurate reflection of the current state of a component.

At the higher levels in the C4 hierarchy, such as *Context* and *Containers*, the models are relevant also to other teams creating other systems. These other teams are mostly interested in the external interfaces offered by components, and in changes to those. To smoothen this cross-team collaboration, it may be valuable to eventually model these levels in a more formal way rather than the informal PlantUML diagrams that are currently used and that may provide limited abilities for automated analysis.

For the same reason, there may be benefits from more formal modeling when connecting the development of embedded software more to systems engineering. The goal is then to integrate the development of the embedded software closer to the other departments in the company, that develop the hardware and electronics of the products. Currently, the documentation effort for new and existing software components has priority and it is not yet a priority to expand these efforts. However, a company-wide Product Lifecycle Management effort is running in parallel, which includes a possibility of cross-domain modeling. Within this initiative there is an opportunity to create an environment of joint modeling for the different software domains such as cloud, handhelds and embedded as well as with the non software domains of mechanics and electronics.

5.2 Feedback from the software engineers (Quantitative)

We sent out a questionnaire survey by email to all (to limit selection bias) software engineers for embedded software at the company. A reminder was sent after 2 weeks. We got 48 responses from the complete sampled population of about 150 software engineers, a response rate of about 30%.

We structured the questionnaire as follows. First, we asked basic demographic questions: years of employment, role, and type of team. The years of employment are important to distinguish the responses from engineers that have experienced the way of working also before the changes were rolled out two years ago. The opinions of the newer employees on the new way of working are still interesting, even when these employees cannot compare their experience to a previous way of working. Second, we included questions on the frequency with which diagrams at the various C4 levels were created, edited, and viewed. Third, we included other questions about experiences working with C4. In particular, we inquired about mental load of using the way of working, the degree of success that engineers had with performing their tasks and the effort in terms of hard work and time spent. We repeated the similar question to ask those engineers familiar with the prior way of working to compare these task load aspects between the two ways of working. Fourth, we included open questions about experienced benefits, challenges, and foreseen challenges in the future. The complete questionnaire instrument is available as supplementary material [13]. To limit threats to construct validity, the questionnaire was created in several iterations with feedback from the architecture team. We now discuss the answers to each of the parts of the questionnaire.

Demographics Most (35) of the respondents have been employed at the company for more than 2 years, and thus have experienced both ways of working. All but two of the respondents are software engineers; one is a product owner and another a software tester. The respondents are almost equally divided among the types of teams: product (24) and platform (23), one respondent replied "other."

Creating/editing/viewing C4 diagrams Before asking their concrete experiences with the C4 diagrams, we are interested to know how often engineers interact with them. Therefore, we asked respondents to grade on a 7-point scale the frequency with which they (i) create new diagrams, (ii) edit existing diagrams, and (iii) view existing diagrams at each of the C4 levels. The responses are aggregated in Figure 1.

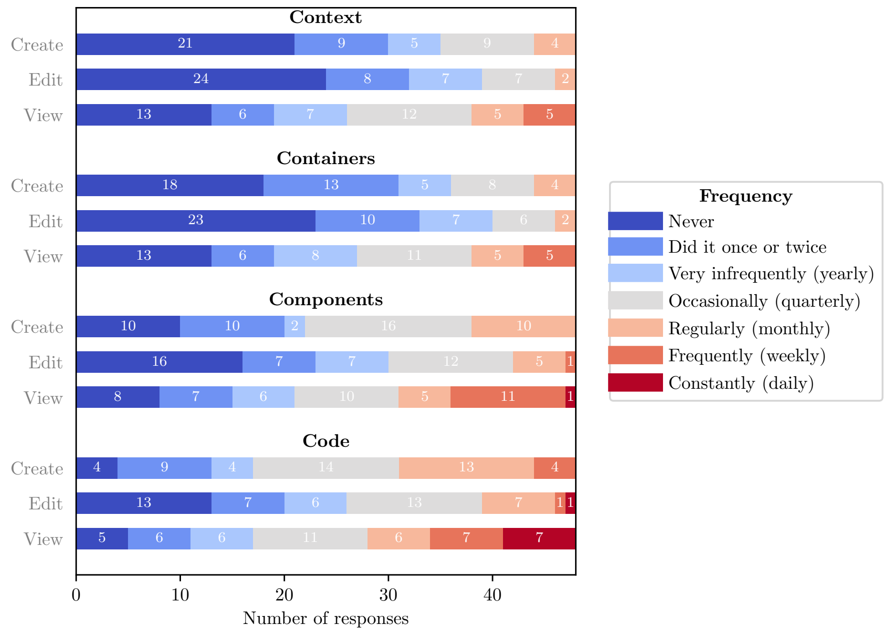

Fig. 1. Summarized overview of frequency of creating, editing, and viewing diagrams at the different C4 levels, as indicated by survey respondents.

Viewing is the most common way to interact with the diagrams, which is expected. Respondents indicate creating new diagrams with slightly higher frequency than editing existing diagrams. Due to the relatively short time since the introduction of the new way of working, the engineers are mostly engaged in creating new documentation for components that are missing it, rather than updating documentation to reflect changes to the implementation. There are a total of 466 features to be documented, but only 42 components currently have a C4 code level document. Hence, most effort at this point is spent on filling this gap by creating new documentation.

A general trend that can be read from Figure 1 is that the engineers most frequently interact with diagrams at the lower C4 levels. The diagrams at the higher levels see very infrequent use, with a third of the respondents even indicating to never view diagrams at context and containers levels. Both these results follow our expectations that software engineers would indeed most often need the information expressed in the code level diagrams. Moreover, this observation shows that the high-level architecture of the system is stable, few changes are needed to its documentation.

Task load of new way of working To elicit experiences with this new way of working, we selected relevant questions from the NASA Task Load Index [14] and adapted them to make them suitable for our context. We asked two questions and for each of them four sub-questions on the same topics. The two questions asked respondents to (i) indicate their experiences with the new way of working, and (ii) indicate their perceived difference with the previous way of working. These two are referred to as 'Absolute' and 'Relative' respectively in Figure 2. For the 'Relative' question, we only include the answers from those respondents that answered the questions and indicated more than 2 years of employment, since they have worked with both approaches. The bold text labels in the figure indicate the four sub-questions, which asked respondents to rate (i) the mental load of working with the C4 model, (ii) how successful they were in accomplishing what they are asked to do, (iii) how hard they had to work to accomplish the required level of performance, and (iv) how much time they spent on accomplishing the documenting tasks using the C4 model.

Figure 2 shows a rather diverse experience with the new way of working. Note that for mental load, hard work, and time effort, lower values are generally more positive than higher values but not necessarily so. Since the previous practice (that the engineers are asked to compared to) for some of the respondents was to not document at all, some respondents will have marked a higher effort. Almost half the respondents indicate a lower mental load, and about a third think it is more difficult to work with the new way of working. Half the respondents indicate being more successful in accomplishing what they are asked to do using the new way of working compared to previously, a quarter think they are less successful. We now look at the answers to the open questions to further interpret these quantitative results.

5.3 Feedback from the software engineers (Qualitative)

To complement the previously presented quantitative data, we included open questions in the questionnaire to gather qualitative data, too. We asked three questions about (i) benefits, (ii) challenges, and (iii) foreseen future challenges of the new way of working. A final question was included to gather any other feedback for the architecture team or the researchers. To those last four questions, we gathered 32, 33, 28, and 19 answers, respectively. We now summarize

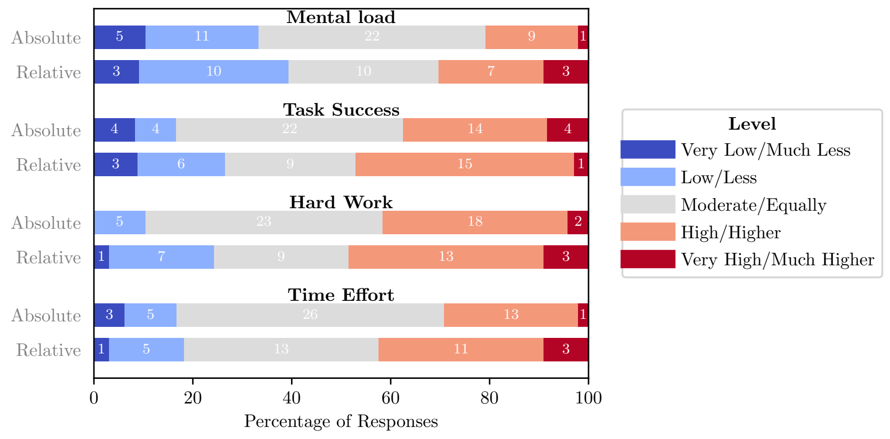

Fig. 2. For the four questions about task load, we are plotting the number of responses to the two questions about the absolute task load and the relative comparison to the previous way of working. The size of the bars indicates the percentage, the numbers inside the bars indicate the number of responses.

the responses to each open question and include some supporting statements from the responses in *italics*.

Experienced benefits A categorized overview of the experienced benefits, and the number of answers they are mentioned in, is shown in Table 1.

Table 1. Categorized benefits as mentioned in the 32 responses, answers have been assigned to one or more categories.

Category of response	#
Documenting structure and well-defined process	15
Documentation located close to the code	13
Lightweight, fast, or otherwise easy to use	7
Improved understanding of implementation	6
No benefits seen	3

Most mentioned benefits (15 of the 32 responses to this question mention this) are in the improved organization of the documentation and an improved process of documenting. Very related to that, having the documentation close to the code is mentioned as a benefit in 13 of the responses. As an example of both, one respondent noted: *"It is very nice to have the architecture work standardized, so you know what to do, and you know where to find the documentation. I like that code and documentation are in the same repository."* The benefit of co-locating documentation and code is mostly seen in a better alignment of the documentation to the code: *"It is a good thing as it ensures alignment of code and documentation. It streamlines the naming and structure of components."*

Related to the specific way of working, 7 respondents explicitly mention ease of use and speed as benefits. In particular the textual notation and inclusion of pictures is appreciated for its simplicity: *"Markdown is great for documentation because you are only allowed to write text, and put in pictures. Nothing fancy, so you have to make clear documentation."* One respondent confirms that it remains important that the new way of working provides a: *"better visualization of what we are trying to achieve."* The learning curve for PlantUML is mentioned but experienced as manageable.

The better organization of the documentation and the process of documenting the architecture also leads to easier understanding of the code, both for existing developers and newcomers, as mentioned in 6 responses. For example, they indicate an improved understanding of code behavior, dependencies, and interfaces. The new common understanding of the requirements for documentation has benefits in both creating and reading documentation: *"We now have a common understanding of the required amount of documentation and a standard to follow for creating/editing it. This makes it much easier to read existing documentation and it is also great to know how much to do when documenting – even if the workload is then a bit bigger. This bigger workload is easily compensated for by the earlier mentioned benefits."* This answer also indicates that the higher workload noticed in Figure 2 is not per se experienced negatively. Overall, these answers align with the experienced benefits as mentioned by the team of architects.

Experienced challenges A categorized overview of the experienced challenges, and the number of times they are mentioned, is shown in Table 2.

Table 2. Categorized challenges as mentioned in the 33 responses, answers have been assigned to one or more categories.

Category of response	#
Learning curve and time spent	11
Limited control over diagram layout	7
High effort and sometimes low value	6
Migrating to new way of working	5
Granularity of C4	2

The most mentioned challenge (in 11 responses to this question) is related to learning the new way of working and taking time to adopt the new process. As with any new way of working, some new learning is required, so this is not a very surprising result. Moreover, one response puts this in challenge perspective of the lack of alternative: *"Learning PlantUML 'coding'... but hey – better than working with, e.g., Visio."* Responses also indicate that respondents experienced good support from the architecture team, created guides, and online resources to adopt the new way of working and learn the PlantUML syntax.

Since PlantUML diagrams are automatically rendered, there is limited control (by design) over the exact graphical layout of the diagrams. This is mentioned as a challenge by 7 respondents and can be exacerbated when the complexity of the diagrams grows: *"Diagram output is hopeless when complexity increases, lot of time spent on getting diagrams being readable"*

Some reported challenges related to the migration to the new way of working and the expected content of the documentation. As mentioned by 5 respondents, one challenge is in knowledge management of the features: *"Creating new C4 is easy but otherwise, knowledge about the feature and its purpose is needed but hard to get from previous owner."* This indicates a challenge of communication between the teams and other stakeholders. It then is also important that the teams document in the same way, to make it easier to understand documentation by anyone. One respondent experiences this challenge: *"Among teams it's very different how C4 is interpreted, hence the C4 artifacts are on very different quality levels."* Lastly, challenges are experienced in maintenance of the new documentation: *"In case of updating an existing feature for a new product, the time to convert to C4 considering backward compatibility is very high."*

Two responses mention challenges related to the granularity of the documentation. One specifically mentions the gap between containers and components, and seeks an intermediate breakdown. There are also some negative responses (6), that indicate experiencing little value of the new way of working, but at the same time a high effort in terms of time spent.

Foreseen future challenges A categorized overview of the anticipated future challenges, and the number of times they are mentioned, is shown in Table 3.

Table 3. Categorized challenges as mentioned in the 28 responses, answers have been assigned to one or more categories.

Category of response	#
Maintenance effort and overhead	8
Quality assurance	6
No foreseen challenges	5
Alignment with feature model	4

Since the new way of working was rolled out two years ago, we considered that there may be challenges that are foreseen by the engineers but not yet encountered. This question anticipates challenges related to maintenance and indeed that was the most frequently recurring theme in the answers (8 mentions). Respondents expect future challenges in keeping the documentation aligned with the implementation in an efficient way: *"Keeping the documentation aligned with the code. However, I feel, that the C4 documentation interacts with the code in* *a way that will make such an alignment easier – or more likely to happen – than it has been in the past.”*

Moreover, 6 responses mention that the quality of documentation needs to be high to ensure future maintenance effort is limited: *“We should make sure everybody is on the same level, and perhaps have a bit more governance around C4 to make sure generated artifacts are on the same (high quality) level.”*

Another aspect of maintenance (mentioned 4 times) is related specifically to the way of working at the company, where also a feature model is created and maintained: *“Alignment with the feature model needs to be done.”* Further challenges are mentioned related to the alignment with the feature model and communication between different teams: *“Communicating design and interfaces may need some work. If one is not present during the inception of the work, it is very difficult to deal with the interface and understand documentation. That is, you need to know the person that wrote the document to use the document for some modules.”*

Some (5) responses do not foresee any challenges, or expect that the new way of working will help them in overcoming them: *“None. Much easier to find documentation and edit – also in the future.”*

Other feedback In the last question, any other feedback could be shared. Some respondents shared their concerns about the approach being of high cost and low value, albeit without much supporting argumentation: *“C4 is a kind of white elephant”*. Others are specifically missing the value of the context and containers levels: *“the two top levels are more or less something we create because it is required, but I have yet to see the benefits of these two levels.”* Some question the need for extensive documentation and find the software components simple enough to be understood without it: *“Only maybe 10% of software components needs documentation; 90% of the components are too simple so there is no need to spend time on making documentation (especially specious C4).”* The structure that the C4 model provides is appreciated under some conditions: *“C4 is a reference. For certain implementation, we may need a ‘C5’ or ‘C6’ or say 8 levels of decomposition to explain everything.”*

6 Lessons learned

In this section, we discuss lessons learned during the process of adopting the new way of working and its evaluation.

6.1 Defining the concept of software component is important

The definition of what a software component comprises has been a crucial enabler for the new way of working. Previously, there was no such stringent definition and consequently, different interpretations existed across teams. This limited the abilities for reuse and made the initial efforts for homogenized documentation more difficult. The software component definition has proven to be instrumental in the refactoring efforts to achieve an architecture exhibiting high cohesion and low coupling. Moreover, the concrete definition of software component was a prerequisite for aligning the code-base with the feature model, the requirements structure, and the other aspects of the system design. Formulating a definition of software component was thus absolutely necessary for the successful adoption of the new way of working.

6.2 Documentation close to code is experienced positively

Keeping documentation close to code is experienced positively across the respondents and architecture team. The specific perceived benefits of this practice are accessibility and an increased ability to keep the documentation synchronized with the source code. These benefits are independent of the C4 model and could thus also be achieved with other paradigms to structure the documentation.

6.3 Drawing is still needed sometimes

The way of working rolled out at the company recommends the use of PlantUML and provides templates and guides to use it in combination with Markdown. However, sometimes PlantUML is not enough to capture everything that the engineers want to express, for example when it remains vital to have complete control over the layout for some diagrams. This is due to semantics that are implicitly captured by, e.g., positioning components close to each other or aligned to indicate local groupings or dependencies cross layers. It is thus crucial that the engineers are enabled to still use informal diagramming to express their designs. The new way of working supports this, and some diagrams are still created using "draw.io". Completely changing this approach into a modeling approach with strict adherence to a modeling language seems not of interest to these engineers, partly because the software engineers themselves do not experience benefits from expressing these diagrams in the form of models and partly because of the convenience of using the informal diagrams they are used to: *"For example, I am plotting a sequence diagram, where I would like to add some blocks of a flow diagram or decision making blocks for the sake of better understanding. But this is not possible to do in C4 diagrams, whereas we can do with office tools."* The C4 model paradigm is notation independent. It was the decision of the architect team to use UML at the lower documentation levels, because it has defined semantics which are particularly well suited for those levels, e.g., class diagrams.

6.4 Benefits and challenges

Working with the C4 model is providing a good structure for the documentation, which the engineers appreciate. In addition, the engineers report benefits from the textual representation, especially as compared to the office tools as were in use previously to create informal diagrams. The engineers also scrutinize the new way of working as they perceive it to cost too much effort for too little perceived value. That is put into perspective by others, who mention the additional effort is worth it due to the better state of the documentation. Our results can thus not be unequivocally attributed to the C4 model, but in large part are related to simply having a more systematic means of documenting the architecture.

Overall, the use of the C4 model thus yielded mixed responses, some criticism citing an increase of effort with little perceived value, and at the same time a majority of respondents indicates they are successful in achieving what they are asked to do with the new way of working. In parallel with the roll-out of the new way of documenting, a large refactoring effort has been initiated to remove architectural technical debt. This refactoring has included some conversion of old documentation and the creation of missing documentation. In some cases, developers have found this work redundant, which may have affected the answers.

As future work, we plan to a follow-up of the questionnaire we performed in this study. Establishment of all the missing documentation is a currently on-going process and doing so for all software components will take more time to complete. We expect that a follow-up in two years would be appropriate to investigate if a higher coverage of the documentation is indeed achieved and what the experiences with the C4 Code level documents are by then.

7 Conclusion

In this paper, we have shared experiences of a company's adoption of a new way of working for documenting their software architecture. We described the reasons to change, the details of their practice, including lightweight modeling following the C4 model for software architecture. Moreover, we describe the crucial importance of the formulation of a definition of software component for the company. Two years after initiating the change in way of working, we performed a questionnaire survey among engineers working on the embedded software at the company. The results indicate that they interact mostly with the diagrams at the lower C4 levels (components and code). Compared to the previous way of working with very little architecture documentation and mostly use of informal diagrams, there is an understandable increase in time effort and hard work reported. This additional effort is not per se bad, as the structure the new approach brings is appreciated, and having the documentation next to the code is also appreciated by the software engineers.

We do not claim that these results are generalizable to all industry contexts, but we hope that this experience report can provide input to practitioners with similar considerations on how they can change their software architecture modeling and documentation practices.

References

1. S. Brown, *Software architecture for developers.* LeanPub, 2013.
2. PTC. Pure Variants. https://www.ptc.com/en/products/pure-variants.
3. R. Jongeling and F. Ciccozzi, “Flexible modeling: a systematic literature review,” *Journal of Object Technology*, vol. 23, no. 3, 2024.
4. A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, “Grand challenges in model-driven engineering: an analysis of the state of the research,” *Software and Systems Modeling*, vol. 19, pp. 5–13, 2020.
5. A. Martini and J. Bosch, “A multiple case study of continuous architecting in large agile companies: Current gaps and the CAFFEA framework,” in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA).* IEEE, 2016, pp. 1–10.
6. F. Tian, P. Liang, and M. A. Babar, “Relationships between software architecture and source code in practice: An exploratory survey and interview,” *Information and Software Technology*, vol. 141, p. 106705, 2022.
7. R. Balzer, “Tolerating inconsistency,” in *Proceedings of the 13th international conference on Software engineering.* IEEE Computer Society Press, 1991, pp. 158–165.
8. P. Stevens, “Maintaining consistency in networks of models: bidirectional transformations in the large,” *Software and Systems Modeling*, vol. 19, no. 1, pp. 39–65, 2020.
9. C. Manteuffel, P. Avgeriou, and R. Hamberg, “An exploratory case study on reusing architecture decisions in software-intensive system projects,” *Journal of Systems and Software*, vol. 144, pp. 60–83, 2018.
10. A. Vázquez-Ingelmo, A. García-Holgado, and F. J. García-Peñalvo, “C4 model in a software engineering subject to ease the comprehension of UML and the software,” in *2020 IEEE Global Engineering Education Conference (EDUCON).* IEEE, 2020, pp. 919–924.
11. I. Reyes. C4 Model. My experience + Example. https://itzareyesmx.medium.com/c4-model-my-experience-example-fbcf50def540.
12. K. Czarnecki and M. Antkiewicz, “Mapping features to models: A template approach based on superimposed variants,” in *International conference on generative programming and component engineering.* Springer, 2005, pp. 422–437.
13. R. Jongeling, “Questionnaire survey instrument used in the paper “adopting the C4 model for lightweight architecture modeling – an experience report”,” Jun 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15736948
14. S. G. Hart and L. E. Staveland, “Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research,” in *Human Mental Workload*, ser. Advances in Psychology, P. A. Hancock and N. Meshkati, Eds. North-Holland, 1988, vol. 52, pp. 139–183.