A design decision documentation technique: Five viewpoints to capture the architectural knowledge

Martijn Sturm
m.j.sturm@students.uu.nl

25 June 2020

Abstract

Software systems have become larger and more complex in recent years. Therefore, it is more costly to extend existing software. A technique to help capture and represent a software system's rationale and history helps understanding and communicating this system. In this paper a technique to document architecture decisions is described and applied on a made up case. Also, a Process-Deliverable Diagram is created to support the use of this technique. To conclude, some findings from an interview with the creator of the technique are presented.

Notice of Originality

I declare that this paper is my own work and that information derived from published or unpublished work of others has been acknowledged in the text and has been explicitly referred to in the list of references. All citations are in the text between quotation marks (“ ”). I am fully aware that violation of these rules can have severe consequences for my study at Utrecht University.

Signed:

Name:
Martijn Sturm

Date:
25-6-2020

Place:
Utrecht

1 Introduction

In this section, the technique from Van Heesch et al. (2012a) is described. The technique was created by Uwe van Heesch, in collaboration with P. Avgeriou and R. Hilliard. Uwe van Heesch worked for the University of Groningen when this paper was written.

This technique can integrate with the viewpoints approach described in P. B. Kruchten (1995) and ISO (2011). First, the components of the framework (the viewpoints) are explained. Thereafter, the procedure of applying the technique is described.

Semantics and notation

Decisions are containing more information than can be represented in a single viewpoint-like model. Therefore, Van Heesch et al. (2012a) propose that multiple viewpoints are necessary to document architecture decisions completely. Note that for most viewpoints, actual views were generated in section 2.

Viewpoints. The **Decision Detail viewpoint** contains all the information of every decision that has been considered separately. This viewpoint serves as a sort of catalog for all decisions. It includes information about relations to other decisions, a description of the problem and the alternative solutions, and historical changes if applicable (Van Heesch et al., 2012a).

The **Decision Relationship viewpoint** is used to visualize the dependencies between the individual decisions. In this viewpoint, all decisions are depicted as rounded rectangles in a single diagram including the name and the decision's state. If there is any relationship between a pair of decisions, these are visualized making use of arcs including a description of the type of relationship (Van Heesch et al., 2012a). Examples of relations are: "caused by", "depends on", and "alternative for".

The **Decision Stakeholder Involvement viewpoint** is the least used viewpoint and hence not described.

In the **Chronological viewpoint**, the system's evolution is visualized. System milestones are the nodes, from which arrows connect every decision made after the milestone in a sequential manner. The last decision before a new milestone connects via an endpoint to that new milestone. Every decision contains a state as well. Hence, state changes can be tracked in time by following the arrows that connect all the decisions (Van Heesch et al., 2012a).

In Van Heesch et al. (2012b), the authors proposed an additional **Decision Forces viewpoint**. A force is defined as any factor that plays a role in the system and the system's environment that enables the system to function (Van Heesch et al., 2012b). These forces influence the architect in choosing for a particular alternative concerning an architectural decision. This factors can be project-specific, such as a project's requirement, but can also be driven by an architect's personal experience or a recommendation from literature. This viewpoint describes the relations between the decisions and the forces that influence the decisions that the architects make (Van Heesch et al., 2012b).

In figure 1 the metamodel of all viewpoints is shown. This metamodel is adapted from the metamodels in Van Heesch et al. (2012a) and Van Heesch et al. (2012b). It is simplified so all viewpoints could be included in a single diagram. For the extended metamodels, refer to the original papers (Van Heesch et al., 2012a, 2012b).

In van Heesch et al. (2017), the authors describe several additions for the already existing viewpoints. For this paper, these additions introduce too much complexity. Please refer to van Heesch et al. (2017) for further details.

Description of procedure

In this section, a description is given about the procedure of the technique.

Development activity and deliverable. This technique can be used to execute the documentation development activity. Documentation is an activity that spans most of the phases of software development. However, this technique focuses on the decisions during the design phase mostly. The deliverable will be a combination of multiple diagrams representing the four viewpoints mentioned earlier. These diagrams undergo an evolution as long as the software system is being further developed. The deliverable will be build from scratch during the design phase of the software development process. After implementation and deployment, the deliverable can serve multiple functions depending on the future of the software system and the point in the software lifecycle that is reached.

Before this technique can be applied during the development of a software system, it is necessary that the requirements for the system have been gathered, and that the relevant stakeholders are known. Subsequently, the technique can be applied according to the steps described below.

Determine relevant viewpoints. First of all, select which of the 5 viewpoints are necessary for the project. This depends on the size, complexity, number of stakeholders, among other factors (Van Heesch et al., 2012a). For most projects it is advised to always include the relationship, forces, and detail viewpoints. Depending on the number of stakeholders involved, a stakeholder viewpoint can be helpful. Ad-

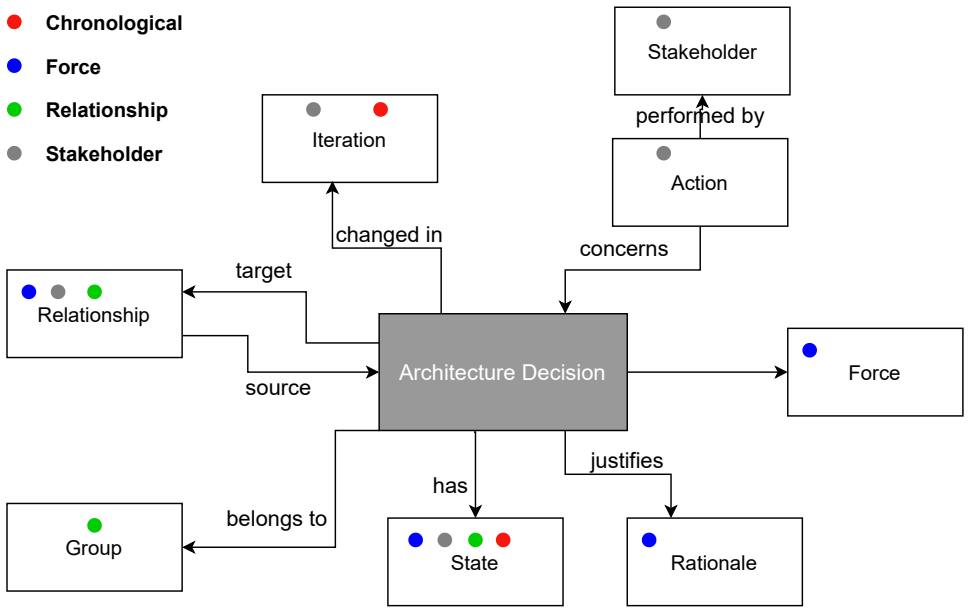

Figure 1: Metamodel of the viewpoints. In which viewpoint the elements are represented is depicted with colored dots. Note that each element is found in the detail viewpoint.

ditionally, if a software system is meant to last for a long time and is expected to undergo a lengthy evolution, the chronology viewpoint should also be used.

Decide which tool will be used. Second, a tool should be selected that is used to perform the documentation. The technique does not include a fixed tool to actually execute the activities and create the deliverable. A minimal supportive tool would be a tool to create diagrams, which are needed for the viewpoints that use entity-relationship based notations (all but the detail viewpoint). Besides, the tool should support drawing tables for the forces and the detail viewpoints. A tool could enable automation of some parts of the activities.

Gather the forces. The next step involves identifying the forces. All forces should be gathered and listed before any decision is made (Van Heesch et al., 2012b). This, so decisions do not have to be reconsidered over and over. Which would be the case if forces were added multiple times when decisions had already been made. The resulting list of forces can subsequently be used for every decision that has to be made. It helps in comparing each alternative's impact on the forces.

Model decisions. After these previously mentioned preparation steps, the actual decision making and modeling of these decisions can begin. In the case study by Van Heesch et al. (2012a), the architects first modeled the relationship viewpoint and established the basis of the decision detail viewpoints. When that backbone was ready, the decision detail viewpoints could be completed further. This was a 'brown field' project, which also resulted in a more effortful modeling of the chronological viewpoint. So in this case, the order of the decisions had to be retrieved in some way from the architects' memory or from the code. In a 'green field' project, it is more convenient to model the decisions in the chronological viewpoint when a single decision has been made. When a decision is made or changed, each view has to be updated accordingly. Then, downstream effects on other decisions of this single change become apparent, and can be acted upon. So this might lead to a cascade of changes and further decisions that have to be made. Note however, that consistently following this path should lead to correctly and completely documented decisions. This process will continue until the development of the software system ends. If the models are needed for cases such as architecture reviews, discussions, or as information source for new project members, the diagrams can be exported and used.

2 Case Example

In this section, a fictional case is described upon which the decision documentation framework will be applied.

Case Description

An academic hospital wants to leverage the power of information systems and combine this with machine learning to improve health care. They envision a software system in which the generated data from all patients and lab tests that are conducted will be stored. This data should be accessible to authorized personnel only. Also, it must be possible to use AI on the data for research and prognosis / diagnosis

with machine learning algorithms. The system must follow the latest and most future-proof architecture principles. Also, it must be extensible, so that future inventions can be implemented in the system. Finally, security and privacy are the most important quality attributes.

Stakeholders. Four main stakeholders are responsible for the system.

- **Architects:** This group of people are responsible for the design of the system.
- **Developers:** This group of people will code, deploy, maintain and keep the system operational.
- **Board of directors:** These persons set the goals of the system and evaluate its functionality.
- **Ethical committee:** This group of people set and test the privacy rules regarding the date that resides in the system.

Initial Requirements. The following requirements were established prior to the design of the system:

- R1: Data should be secure in organization
- R2: It must be possible to authorize employees to different parts of the system
- R3: Data should be anonymized to all researchers that are no docters
- R4: It must be possible to use multiple technologies and frameworks to interact with the data

Procedure

Relevant viewpoints. First, the architects together with the board of directors decide that all decision viewpoints will be used in the system. So the detail, relationship, chronological, stakeholder, and forces viewpoint will all be used. This because the system is expected to play a major role and will continue to evolve over many years. The viewpoint that is least important and hence considered to be omitted first is the stakeholder viewpoint. However, the board of directors takes into account that the involved stakeholders might change over the coming years. Also they see the benefit from documenting personal responsibilities. They think that this might lead to more responsible choices made by all stakeholders and thus improving the systems quality. Nevertheless, we omit the stakeholder viewpoint in this case, because it leads to unnecessary complexity.

Documentation tool. The architects decide that as tool, they will use a modeling tool to create diagrams. They will use the organisation's document management system to collaborate on the diagrams and other deliverables.

Gather forces. The following forces were identified. Note that this is an arbitrary set of forces that just function as example to showcase the decision viewpoints.

- The system is only accessible from within the hospital's network
- Authorization of users
- Logging of user interactions with data
- Extensibility to the latest scientific frameworks
- Potential to grow the system in yet unknown directions

Iteration 1. The architects first make a decision about the general architecture of the system. They decided to go for a microservices architecture, since this alternative fulfills the requirements better than the monolithic architecture (Table 1). Then they decide that they favour Java over Python as programming language to develop the system in. The trade-off of both decisions in the forces view is shown in Table 2. Then, other decisions that build on top of the decisions above, are made. These are visualized in figures 2 and 3

After all necessary functionality is implemented, they decide to test this first iteration and call it Demo 1. Some departments in the hospital are asked to assist in testing. When this is done, the first iteration can be marked as finished. This is also shown in figure 2.

Table 1: Detail decision for Micorservice

Name	Microservices
Current State	Decided
Decision Group	Architectural patterns
Problem	We need to choose the best possible main architecture for the system
Decision	A microservices architecture is chosen
Alternatives	Monolith
Arguments	A microservice architecture is better extensible, scalable and maintainable.
Related decisions	This <<depends on>>Both Python and Java REST APIs <<depends on>>this
Related requirements	R4: Multiple technologies R5: Extensibility

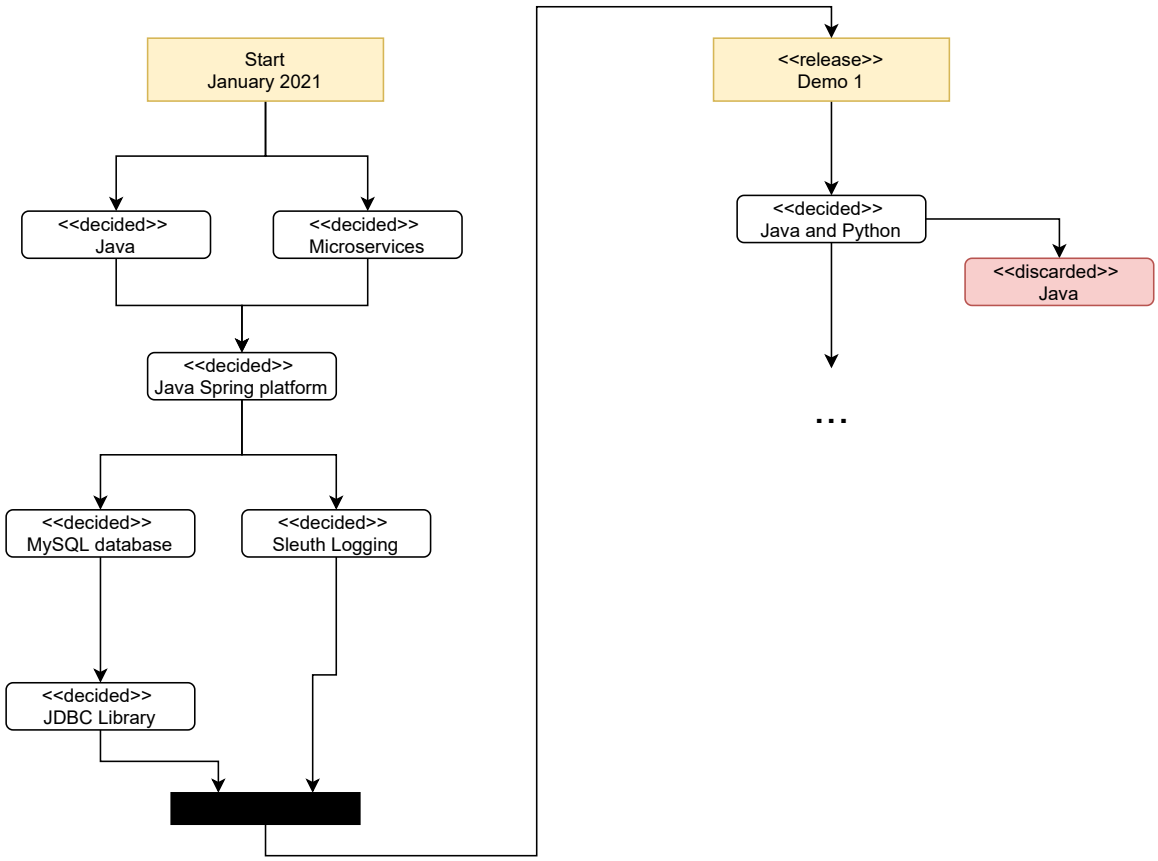

Figure 2: Chronology view. This is the Chronology viewpoint after 2 iterations. The yellow bricks denote milestones. All rounded rectangles are decisions

Table 2: Forces view. Not all forces have to have an impact on each decision. If there is no impact, the corresponding cells are left blank. Pluses represent positive impact, minuses represent negative impact

		General architecture		Programming language	
		Microservices	Monolithic	Python	Java
		<decided>	<discarded>	<discarded>	<decided>
code	Description				
r1	Authorization			-	+
r2	User logging			+	++
r4	Scientific frameworks	+	-	++	-
r4	Growing potential	++	- -	-	++
r1	Network access				

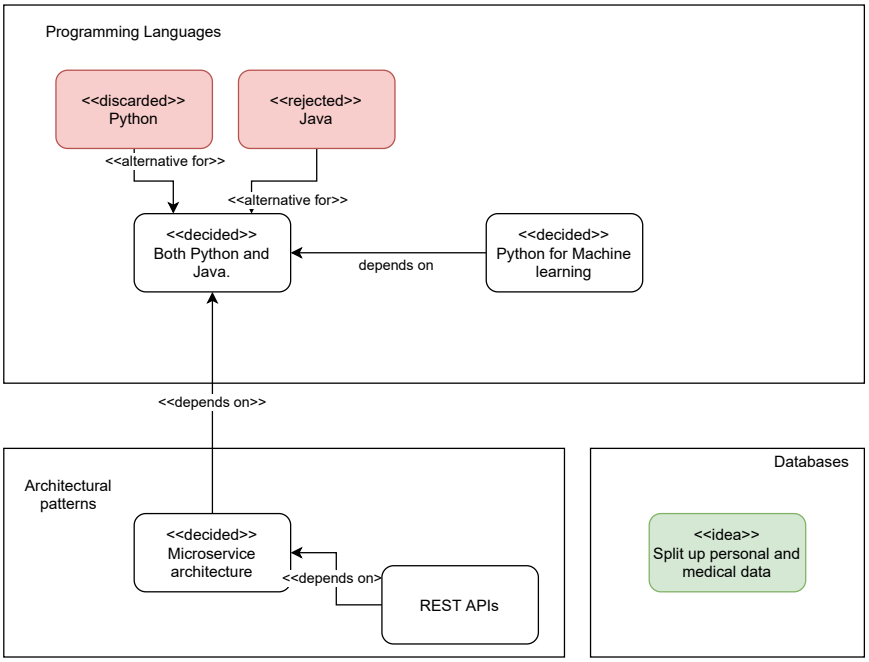

Figure 3: Relationship view. Every rounded rectangle represents a decision. The status is displayed above the decision's name. Decision groups are the boxes surround the decisions.

Iteration 2. Multiple departments have researchers investigating machine learning algorithms to predict disease outcomes for patients based on the patient data. Now every machine learning project is done separately. A platform that enables researchers to collaborate and use data from different departments within the hospital should allow them to do better research. The data should be kept anonymized for the researchers during development of their algorithms however. Contrary, its patient information should be accessible to the doctors that use the algorithm during prediction time.

So now the architects need to decide what to use for the machine learning platform. They decide that Python is needed, because it is the most popular programming language for machine learning. This conflicts with the decision they made earlier to only use Java in the system. Hence, this decision has to be changed. These changes are visible in figures 2 and 3.

Deliverable

Above, the decisions that were made during increment 1 and 2 are described. These decisions were modeled in figures 2 and 3 and in tables 1 and 2. All the viewpoints together form the deliverable of the technique.

3 Process-Deliverable Diagram

In this section, the decision documentation framework technique (Van Heesch et al., 2012a, 2012b) is described and explicated making use of the Process-Deliverable Diagram (PDD) notation. This results in diagrams representing the process and the deliverables (van de Weerd & Brinkkemper, 2009).

The technique is applied to documentation in software development projects. Actors involved in applying this technique are managers, architects, and developers. The architecture and documentation activities are solely performed by architects and developers. For setting up of the documentation process, managers can be involved as well. Section 3.1 describes the technique at the most coarse-grained level of detail. In section 3.2, the process of updating the views per viewpoint will be described.

3.1 Process level

This section describes the most coarse-grained level of the decision documentation framework technique. The PDD for this level is shown in Figure 4. The activities and concepts are described in Tables 3 and 4 respectively.

This process starts when a new architecture decision needs to be made. Then, the activities described in the subsections will be performed. The process ends when the architecture decision is documented satisfactorily according to the architects. The process can be split up in the preparation, decision, update documentation, and monitoring effects activities (Table 3).

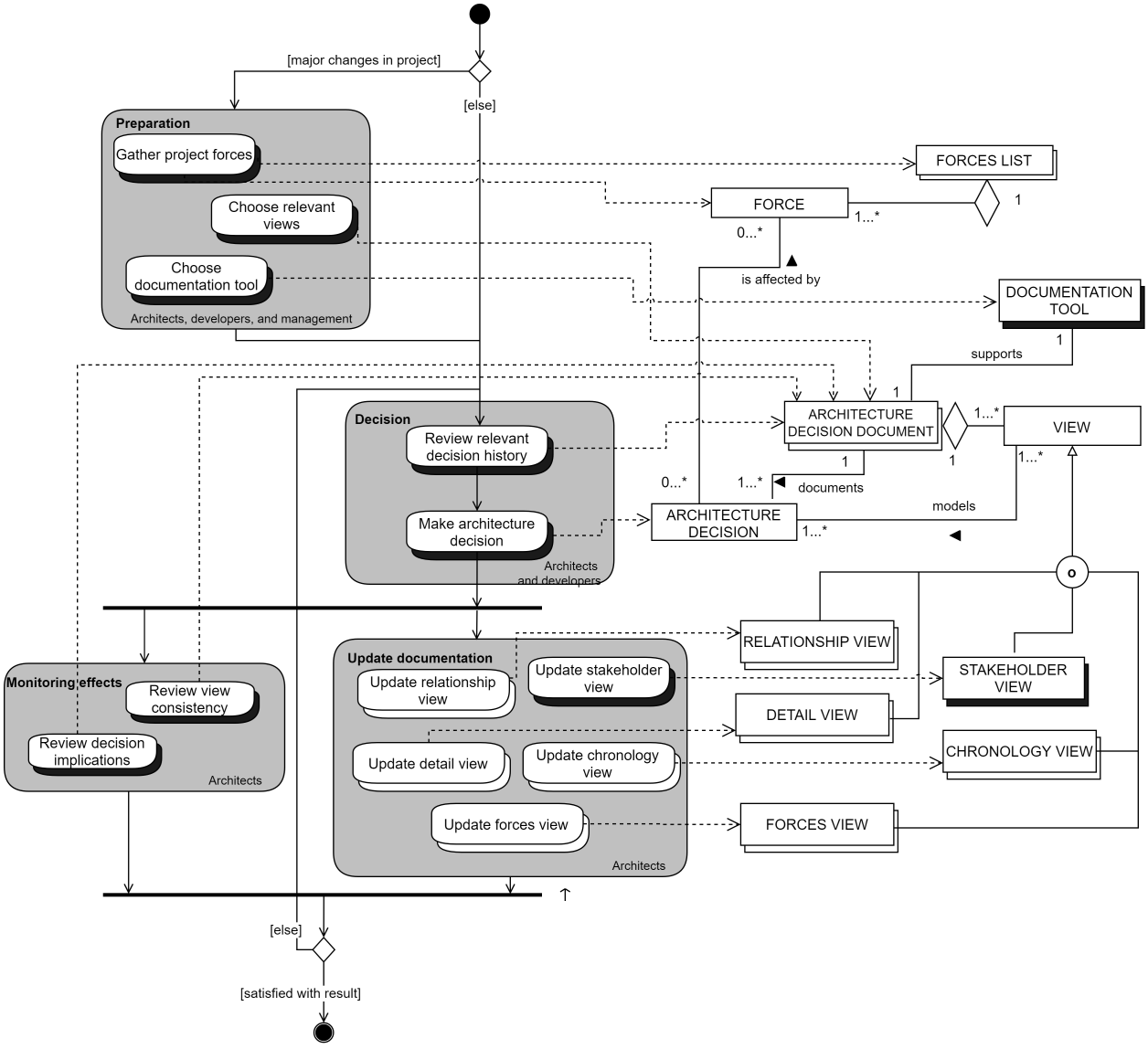

Figure 4: The PDD for the process of decision documentation

Table 3: Activity table for the Architecture decision documentation process

Activity	Sub activity	Description
Preparation	Gather project forces	This activity makes sure that all relevant forces that need to be considered during making of ARCHITECTURE DECISIONs are listed and agreed upon by several stakeholders.
	Choose relevant views	The views that will provide valuable knowledge during this project need to be determined. Which views are relevant can depend on the project's size, the number of people involved, among other factors.
	Choose documentation tool	The tool to make all diagrams, tables, and other content that constitute the ARCHITECTURE DECISION DOCUMENT need to be agreed upon. Working with a single tool makes sure that all VIEWs of the ARCHITECTURE DECISION DOCUMENT can be changed by every employee, and possibly hyperlinked.
Decision	Review relevant decision history	Before being able to make an ARCHITECTURE DECISION, the actors should be up-to-date with the software system's current architecture. Therefore, the actors should review the decision history that is relevant for the decision under consideration. This, so that the consequences of the decision will be compatible with the current software system.
	Make architecture decision	The actual ARCHITECTURE DECISION has to be made before documentation can start. This is a very complex activity. Its details are out of scope of this paper.
Update documentation	Update relationship view	In this activity, the RELATIONSHIP VIEW is updated according to the decision that is made / changed. The details concerning this activity are shown in Fig. 6
	Update stakeholder view	In this activity, the STAKEHOLDER VIEW is updated according to the decision that is made / changed. This view is not described in further detail, since it is the least used view.
	Update detail view	In this activity, the DETAIL VIEW is updated according to the decision that is made / changed. The details concerning this activity are shown in Fig. 5
	Update chronology view	In this activity, the CHRONOLOGY VIEW is updated according to the decision that is made / changed. The details concerning this activity are shown in Fig. 8
	Update forces view	In this activity, the FORCES VIEW is updated according to the decision that is made / changed. The details concerning this activity are shown in Fig. 7
Monitoring effects	Review view consistency	Parallel to the update documentation activity, the effects of the decision on the entire software system need to be monitored. In this sub-activity, the consistency among the views is monitored. Inconsistencies should be addressed and fixed.
	Review decision implications	Also, the implications that the proposed ARCHITECTURE DECISION has on the architecture have to be reviewed. This can become apparent while the decision is modeled in the VIEWs.

Table 4: Concept table for the Architecture decision documentation process

Concept	Description
FORCE	A FORCE affects the ARCHITECTURE DECISION that is made by architects. The FORCEs are collected in a FORCES LIST. (Van Heesch et al., 2012b)
FORCES LIST	All forces that are known to influence the type of software system at hand should be collected and listed in this FORCES LIST. (Van Heesch et al., 2012b)
DOCUMENTATION TOOL	A DOCUMENTATION TOOL will be used by all developers and architects to document the decisions in the VIEWs.
ARCHITECTURE DECISION DOCUMENT	This document contains all selected VIEWs that are created and updated during the software development process. This can be for example a repository containing files representing the VIEWs.
ARCHITECTURE DECISION	A software system's architecture is determined by all ARCHITECTURE DECISIONs that ware taken to create the system.(P. Kruchten, 2004)
VIEW	A VIEW captures certain aspects of an ARCHITECTURE DECISION from a VIEW specific perspective. Multiple VIEWs together make up the ARCHITECTURE DECISION DOCUMENT. (P. B. Kruchten, 1995)
RELATIONSHIP VIEW	Is used to document relationships between ARCHITECTURE DECISIONs. (Van Heesch et al., 2012a)
DETAIL VIEW	Describes the rationale on which an ARCHITECTURE DECISION is based. Also includes other decision specific information.(Van Heesch et al., 2012a)
FORCES VIEW	Describes the relations between stakeholder concerns, forces, and ARCHITECTURE DECISIONs. (Van Heesch et al., 2012a)
CHRONOLOGY VIEW	Describes decisions and their order from a chronological perspective. (Van Heesch et al., 2012a)
STAKEHOLDER VIEW	Describes the interactions between stakeholders and ARCHITECTURE DECISIONs. (Van Heesch et al., 2012a)

3.2 View level

In this section, the PDDs for the four views are displayed in more detail. The detail view is shown in figure 5 and tables 5 and 6. The relationship view is shown in figure 6 and tables 7 and 8. The forces view is shown in figure 7 and tables 9 and 10. The chronology view is shown in figure 8 and tables 11 and 12. The views are split up in separate views to improve readability. Nonetheless, there is some overlap of information between the individual PDDs. For example, in some view PDDs, the technique requires the decision's state to be included in the diagrams. This needs to be consistent between each individual view. Activities that require this type of consistencies are marked with a numbered superscript.

Another thing to note is that the concept DECISION on this level is not exactly the same as the concept ARCHITECTURE DECISION on the top level. Here, a DECISION represents the subset of information from an ARCHITECTURE DECISION that is documented in the particular view. Contrary, the ARCHITECTURE DECISION consists of all the information and influences that are experienced by architects (P. Kruchten, 2004). In other words, the DECISION is an view-specific abstraction of the ARCHITECTURE DECISION.

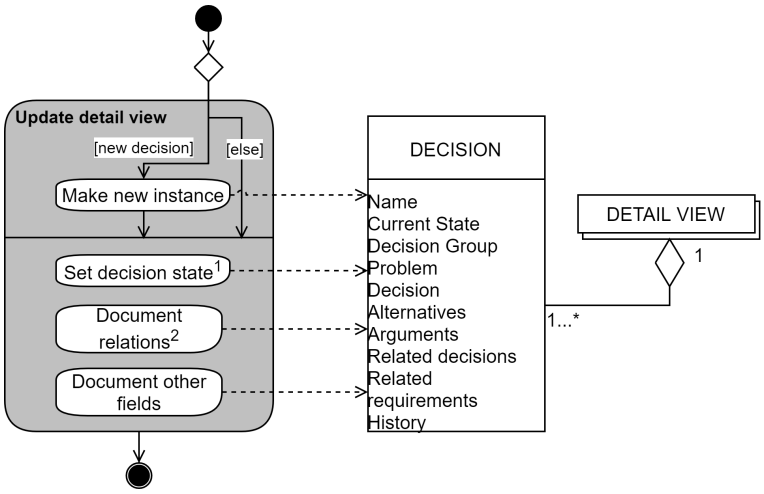

Figure 5: The PDD for updating the Detail View

Table 5: Activity table for the Detail view

Activity	Sub activity	Description
Updates detail view	Make new instance	If the current DECISION is a new one, a new instance of DECISION in the DETAIL VIEW needs to be created.
	Set decision state	Set the DECISION's state in the document.
	Document relations	Document to which other DECISIONs this DECISION has relationships. Make sure this is consistent with the other VIEWs.
	Document other fields	Set all other fields of this DECISION instance. These are shown as the DECISION concept's attributes.

Table 6: Concept table for the Detail view

Concept	Description
DETAIL VIEW	A VIEW that contains details for every DECISION. It can be a hyperlinked document for example. (Van Heesch et al., 2012a)
DECISION	Contains all details of a single DECISION. Can be structured as a list, or a table. It contains the following properties: The Name should resemble the decision that is taken. The Decision Group can be read from the RELATIONSHIP VIEW. The Problem describes the context and need for the DECISION. The Alternatives describes the option to solve the Problem. The Arguments contain the rationale for each Alternative. Related decisions can be copied from the RELATIONSHIP VIEW. Related requirements need to be documented. History contains all the changes of the DECISION's State. (Van Heesch et al., 2012a)

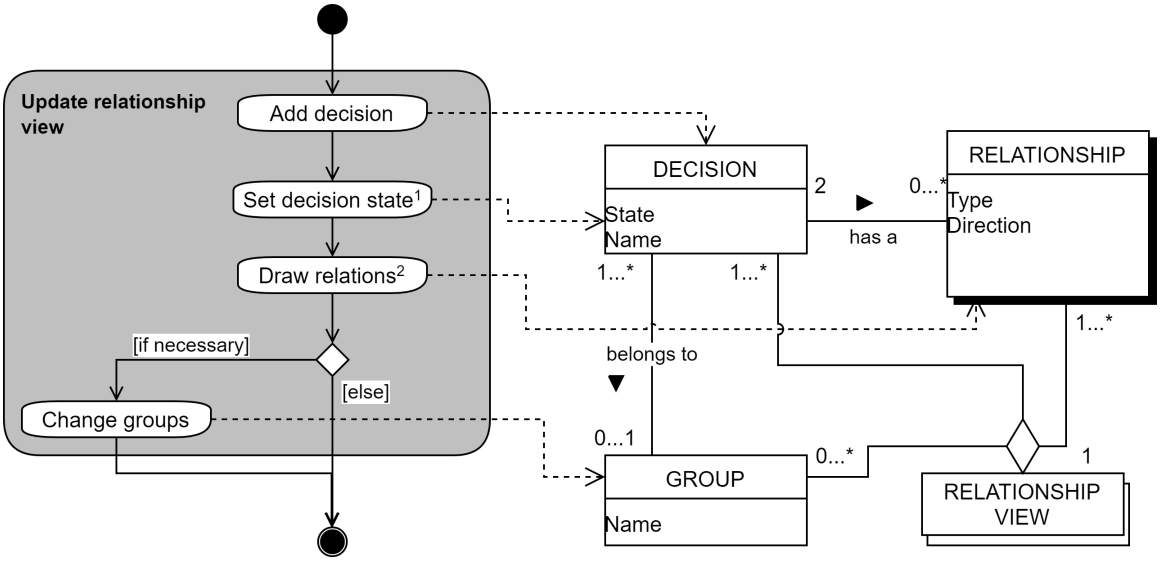

Figure 6: The PDD for updating the Relationship View

Table 7: Activity table for the Relationship view

Activity	Sub activity	Description
Update relationship view	Add decision	A DECISION is added to this view by creating a new box containing the DECISION's name.
	Set decision state	The DECISION's state is added to the box.
	Draw relations	RELATIONSHIPs between the new DECISION and existing DECISIONs are drawn making use of arcs that contain a description of the RELATIONSHIP's type and the direction of that type.
	Change groups	The GROUPs should be altered if the added DECISION has resulted in a need for change. For example, a new category could become apparent as result of the new DECISION.

Table 8: Concept table for the Relationship view

Concept	Description
RELATIONSHIP VIEW	A VIEW that documents the RELATIONSHIPs between DECISIONs. A diagram can be used to model these interaction, which are stored in the RELATIONSHIP VIEW. (Van Heesch et al., 2012a)
RELATIONSHIP	Connects a DECISION to an other DECISION. This has a type, and a direction indicating the effect the DECISIONs have on each other. (Van Heesch et al., 2012a)
DECISION	A representation of an ARCHITECTURAL DECISION simplified for the RELATIONSHIP VIEW. It is depicted as a box in the diagram, contain the DECISION's name and state. (Van Heesch et al., 2012a)
GROUP	A category that contains DECISIONs. This is visualized by colouring or big boxes including (multiple) DECISION boxes in the diagram. (Van Heesch et al., 2012a)

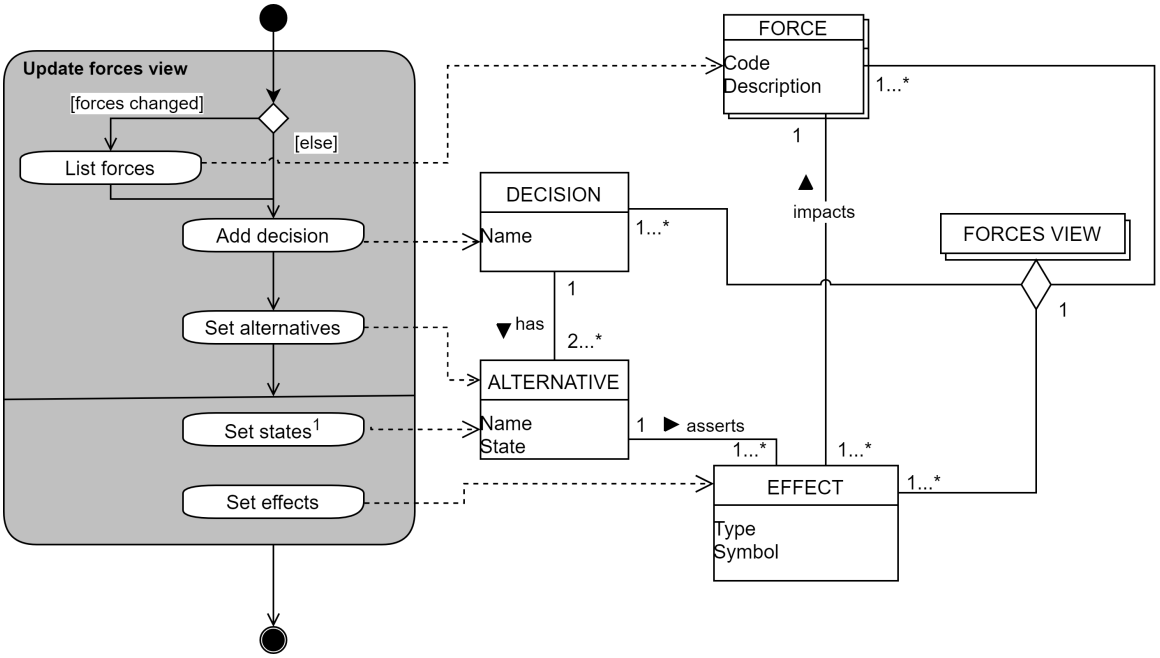

Figure 7: The PDD for updating the Forces View

Table 9: Activity table for the Forces view

Activity	Sub activity	Description
Updates forces view	List forces	If the FORCEs have changed since the last time the FORCES VIEW was updated, then the FORCEs depicted as rows in the table should be updated.
	Add decision	Add the DECISION as a new double column to the table.
	Set alternatives	Add the ALTERNATIVEs and their states as single columns under the DECISION column.
	Set states	Add the STATE of each ALTERNATIVE to the column name.
	Set effects	For every cell in the table, put the corresponding EFFECT that the ALTERNATIVE has on the corresponding FORCE.

Table 10: Concept table for the Forces view

Concept	Description
FORCES VIEW	A VIEW that documents the DECISION's ALTERNATIVEs and the impact on the project's FORCEs. It is depicted as a table with in the rows the FORCEs, and in the columns the DECISIONs and their ALTERNATIVEs. (Van Heesch et al., 2012b)
FORCE	A part of the problem that should be solved by a DECISION. (Van Heesch et al., 2012b)
DECISION	A representation of an ARCHITECTURAL DECISION simplified for the FORCES VIEW. It is depicted in a row, in which the ALTERNATIVEs coresponding to this DECISION are merged.
ALTERNATIVE	One of the proposed solutions concerning a DECISION. It is depicted in the columns, resulting in one column per ALTERNATIVE.(Van Heesch et al., 2012b)
EFFECT	This depict the impact of a single ALTERNATIVE on a single FORCE. It is placed in the cell on the axis of the coresponding column and row. In the table, the EFFECT's symbol is used, which is explained somewhere in the FORCES VIEW as a legend for example. (Van Heesch et al., 2012b)

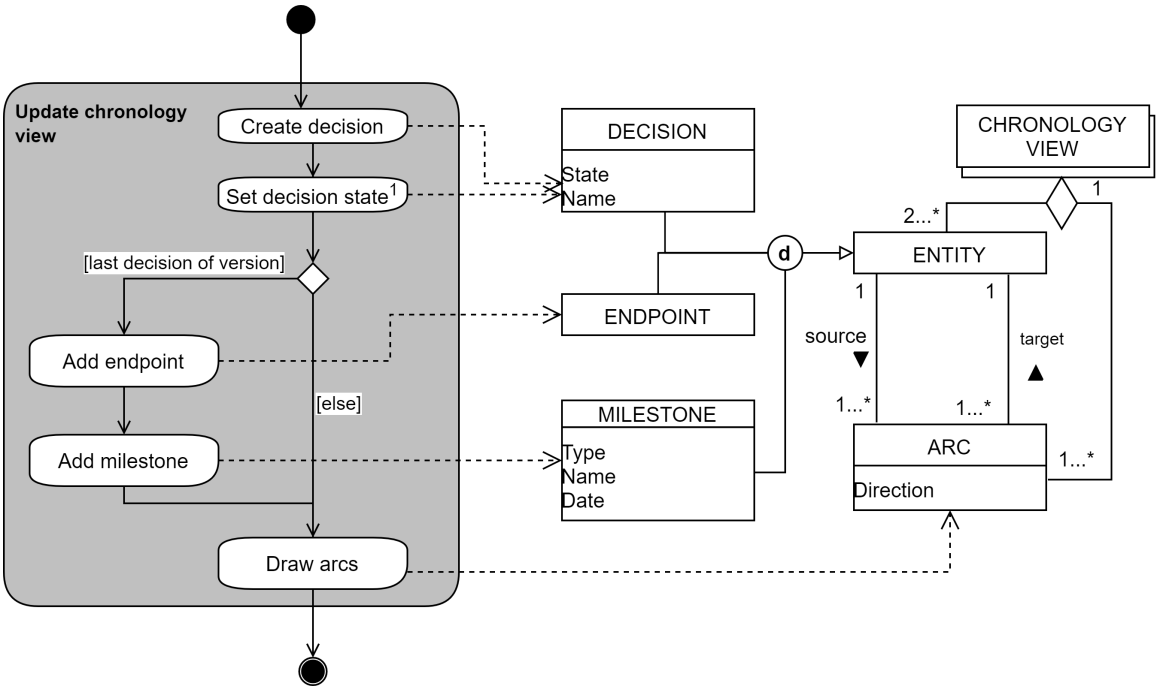

Figure 8: The PDD for the Chronology View

Table 11: Activity table for the Chronology view

Activity	Sub activity	Description
Updates chronology view	Create decision	Create a new DECISION which is modeled as a box.
	Set decision state	Add the DECISION's state to the DECISION ENTITY.
	Add endpoint	If this current DECISION is the last one belonging to a particular version of the software system, add an ENDPOINT to which all the latest DECISIONs connect.
	Add milestone	Add a new MILESTONE, which marks the start of a new version.
	Draw arcs	Connect all ENTITYs to each other making use of directed ARCS.

Table 12: Concept table for the Chronology view

Concept	Description
CHRONOLOGY VIEW	A VIEW that contains the history of DECISIONs for a project. it is depicted as a diagram consisting of ENTITYs visualized as boxes, which are connected by ARCs. An ENTITY earlier in time is the source of another ENTITY. (Van Heesch et al., 2012a)
ENTITY	An event or bundle of information modeled as a box in the CHRONOLOGY VIEW.
ARC	A connection between two ENTITYs. It contains a direction, point towards the most recent ENTITY. This concept represents the chronology dimension of the CHRONOLOGY VIEW
ENDPOINT	A type of ENTITY. All ARCs originating from the last ENTITYs of a particular version of the software project connect to this. (Van Heesch et al., 2012a)
MILESTONE	A type of ENTITY. All ARCs to ENTITYs in a new version of the software project originate from this. (Van Heesch et al., 2012a)
DECISION	A type of ENTITY. Contains all details of a single DECISION. Can be structured as a list, or a table.

4 Related Literature

Software architecture. Software architecting is the activity that connects the requirements of a software system, with the actual implementation of the system. This is achieved by three main steps of architecting. First the requirements are analyzed. Second, solutions are proposed to fulfill the requirements. Finally, evaluation takes place to check whether the chosen solutions fulfill the requirements (Hofmeister et al., 2007). When the evaluation result is considered positive, the architecture can be implemented, resulting in a solution. Thereby, software architecture has bridged the gap between the problem-space (requirements) and the solution space (the software system).

Software architecture is a complex topic. To support architects, different guidelines were proposed (ISO, 2011; P. B. Kruchten, 1995; Perry & Wolf, 1992). These help in maintaining consistency, and suggest several perspectives to use to model a software system.

Software development suffers from architectural erosion. Architectural erosion or design erosion (hereafter called erosion) is a phenomenon appearing during software systems development (Van Gurp & Bosch, 2002). Erosion leads to a decrease of maintainability of the system in the long run. Typically, erosion is caused by a couple of factors. One is the lack of traceability of architectural decisions[1]. Also, the iterative nature of software development methods and their emphasis on increments over documentation contributes. Additionally, the increase of a system's complexity increases erosion (Van Gurp & Bosch, 2002).

Erosion can lead to a system being redeveloped from scratch. This, in turn, can lead to very high costs in resources and time being invested (Van Gurp & Bosch, 2002). Although it is possible to counter erosion via certain strategies, a more fundamental approach would be to fix the cause of erosion (Van Gurp & Bosch, 2002). We build towards this purpose in the upcoming sections.

Architectural Knowledge. De Boer et al. (2007) describe the term Architectural Knowledge (AK) as the knowledge about an architecture. This AK resides for a large part in the minds of involved architects. Therefore, it can be hard to preserve and communicate it. De Boer et al. (2007) propose an AK model, which consists of all elements that they consider to be part of a system's architecture. The modelled AK has several strategic purposes for an organization. It can be used to share information between stakeholders, help in keeping the system compliant with set architectural rules, help auditors to discover the system's architecture, and improve the traceability of AK (De Boer et al., 2007). Design decisions are one element in the AK model, and can be used to enforce, reflect, and inform stakeholders (De Boer et al., 2007).

Design decisions help preserve relevant information. The AK concept described above could potentially solve the problem of the erosion caused by a lack of traceability of design decisions. A technique that enables documentation of these design decisions, and hence improves their traceability, has the potential to decrease a system's erosion over time. P. Kruchten et al. (2009) and Bosch (2004) both call for more focus on design decisions in software architecture. Bosch (2004) proposes that design decisions should be 'first class citizens'. This is in contrast with previous documentation practices, in which the structures instead of the rationales of a system are represented (ISO, 2011; P. B. Kruchten, 1995). Documenting decisions can have multiple benefits. It has the potential to show changes over time, show rationales and options that were considered, and improve traceability of constraints on the system. Also, it can combine well with agile development methods. Because development can start when a decision is taken if the implications and constrains described in the decision are adhered to during development (Bosch, 2004).

Defining design decisions. Design decisions are defined as follows: They should dictate the structure of the architecture. They may define rules and constraints for development and the architecture's components. Also, they should include a rationale for the decision (Bosch, 2004). Additionally, P. Kruchten (2004) defined some properties that design decisions should have, which are the following. The decision's scope should be constrained by defining topics to which the decisions belong. Additionally, the decisions should have a state, and decisions should have relations to decisions and the architecture's other artifacts (P. Kruchten, 2004).

Decision modeling techniques

In the above sections, software architecture is described as the link between the problem space (*i.e.* requirements) and the solution space (*i.e.* software). These three concepts can be continuously evolving over time and often grow in complexity. Also, design decisions contain all the information about why certain structures were implemented in the architecture over alternative structures. Hence, to capture and preserve all this information, a technique is needed that models the architecture decisions. Preferably, this technique integrates with already existing architecture documenting techniques such as the viewpoints approach (ISO, 2011; P. B. Kruchten, 1995).

[1] 'Architectural decision' and 'design decision' are interchangeably used hereafter. They refer to the same concept.

Several papers have proposed ways to document design decisions. Below are some examples described.

Decision templates. The paper of De Boer et al. (2007) stated that making decisions is ranking alternative solutions to a problem. They also applied techniques from Tyree and Akerman (2005) and P. Kruchten (2004) to validate their conceptual model (De Boer et al., 2007).

Tyree and Akerman (2005) propose a template for architectural decisions. For each decision multiple fields are documented: Administrative fields such as the issue, decision status and the group the decision belongs to. Input factors that influence the decision made, and consequences that the decision has on the system. Also relations between components of the architecture and notes about the process of the decision are registered (Tyree & Akerman, 2005). When a decision has been made, the consequences need to be analyzed. Subsequently the architects should determine if the consequences should be converted into decisions as well (Tyree & Akerman, 2005).

P. Kruchten (2004) defines an ontology for architecture decisions rather than providing a technique to document them.

Tools. Tang et al. (2010) use a framework to compare tools that can support AK management. Functionality that the evaluated tools provide include traceability, relationship modeling, visualisation, and management.

Extend decision templates. The main paper that this article is about (Van Heesch et al., 2012a) has a component (decision detail viewpoint) that is inspired by the decision template from Tyree and Akerman (2005). Consequently, a decision template can be seen as part of an architecture documentation framework. But a decision template on its own does not provide easy visualization of decision's dependencies and relationships (Van Heesch et al., 2012a). Especially for larger projects, it can be expected that decision-template-only approaches will perform less in communicating an architecture's structures and rationales to stakeholders that are unfamiliar with the system. A conclusion from a case study was that decision templates were considered less effective and beneficial by students than the decision forces viewpoint (Van Heesch et al., 2012b).

Annotations. Another option to document design decisions are annotations. Annotations are very lightweight, easy to implement, but therefore lack a certain robustness in documentation capabilities. They can be used to create the decision information according to the viewpoints. Though, they would benefit from some additional formalization and logic besides the annotation to enforce consistency and allow for (automated) analysis (Van Heesch et al., 2012a). For an example of an annotation technique, refer to Liang et al. (2009)

5 Findings

In this section, some additional findings and future perspectives are presented. These were based on an interview with Uwe van Heesch, creator of the technique. The findings that are presented here are the writer's interpretations of this interview. Therefore, the writer carries the full accountability. Note that most findings are based on experience and opinions, these are not backed by research data.

The subjects addressed here are related to the use of decision documentation in practice in industry. First, the significance of the technique and factors that influence its usage are described. This is followed by how it can be used. Finally, some future developments related to documentations in software architecture are discussed.

Significance of technique

Industry recognizes need for decision documentation. Nowadays, many huge software systems exist. These might be the result of years of development, and hence years of stacking decisions on top of each other. Parties in industry recognize the lack of transparency of historical decisions that were made. This results in some practical problems. First, it is not known why decisions were made, so people are reluctant to change them. Nevertheless, the decisions are revoked and changed. Which in turn can lead to reappearance of problems that were tackled by the decision that was revoked. Both cases can lead to a decrease in product quality. So in short, not documenting decisions can lead to difficulties in maintenance and development.

Release cycles of software systems have become much shorter in recent years. This also contributes to the need of a lightweight method for decision documentation.

Minimize and focus the effort. So the need for decision documentation is recognized. However, this does not mean that everything that passes your mind should be documented. Documentation is seen by most people as a burden, and hence, is skipped frequently. A factor that could contribute is that documenting is often performed after the to-be-documented activity is completed. Therefore, focus should be on documenting information that is useful, keep the information as narrow as possible, and do it during the activity.

Usefulness of decision documentation depends on project factors. The usefulness and applicability of a decision documentation technique for a par-

ticular software system project depends on some factors.

The first factor is the size of the software system to be designed. When systems increase in size, the number of decisions to get to that size logically increases. Thereby, more decisions build on historical decisions, resulting in an increased complexity. The complexity can increase to an extent that it is hard to oversee all the implications based only on memory of architects.

Another factor is the expected durability of the software system. IT frameworks come and go. Frameworks that are rising in usage right now could be superseded by others in the future. When a software system is expected to have a long life, it will be there to experience the rise and fall of many technologies and frameworks. To be more flexible in applying the latest technological developments, it is necessary to know why for particular decisions certain frameworks were chosen. If so, decisions can be replaced in a more informed way. Concluding, decision documentation is more important for software systems with longer lifespans.

Implementation in Practice

This section describes the place of the framework in the process of architecting a software system. Some practical measures are discussed. Additionally, the usefulness of certain parts of the framework are described.

Make decision documentation part of the development process. If decision documentation is identified as useful for a project, it should be performed consistently. Therefore, it should be included as an activity in the main development process used by the organization. Architects may be responsible to provide foundations and resources to realize decision documentation.

The decision documentation framework is a solid basis. There are a few organizations that use the decision framework. However, it is more likely that implemented decision documentation uses a variation of the technique. This is a good way to use the framework. You could see it as a toolbox from which certain aspects can be selected and implemented. It is not meant as a rigid structure. Organizations can adapt it so it fits their goals and projects better.

Use the relationship view for a quick overview. The relationship view is evaluated as a perspective that helps in obtaining an overview of decisions. For example, a new project team member is trying to get a grasp of the foundations of the product that is being build. A way to inform yourself about the technology stack being used would be reading the documentation. An easier way would be looking at a relationship view diagram, where these technologies are grouped together. Besides, you can also explore which further implications these decisions have on other decisions.

Forces view can be used while making and evaluating decisions. The forces viewpoint is used in industry during the architecture synthesis and evaluation. First, the forces that are expected to influence the decision are selected. Based on this, alternatives are weighted and the best one is selected. This is followed by an evaluation of the proposed alternative, which involves checking whether the other forces are compliant with the proposed alternative. If this is the case, the decision is concluded. If not so, the alternative weighting can be repeated with the previously violated forces included this time.

Future directions

Since the publication of the first paper about the decision documentation framework (Van Heesch et al., 2012a), some years have passed. With this, also the technologies and methods in software development have evolved. Here, some new developments concerning documentation are addressed.

Documentation is partly automated. Continuous software development aims at using specification in files for evoking deployment of systems for example. This is achieved by making use of provisioning tools. With these specifications, documentation can easily be generated automatically. There is a caveat here however. The factual information is included in the specification, but the rationale is not. So only the structure and the decision alternative is captured. So to be complete, there is still a need to do manual documentation. Otherwise, rationale, and alternatives will not be traceable.

In the future, there might be a solution for this, however. Data science techniques could potentially deduce information about rationales from architecture data. For example, if many software systems use a combination of framework A with framework B, the deduction could be that these frameworks are a good combination. Though there is a lot to investigate further. So we can not give precise estimations about how valuable this information would be for this application.

6 Conclusion

In this paper, the technique from Van Heesch et al. (2012a) was demonstrated making use of a made up case. Also, a PDD was created, to support for this documentation activity in software development. As future research directions, it would be interesting to investigate what a tool supporting this technique should look like to be an effective tool. Also, research into data mining techniques for architecture decision documentation could lead to a reduced effort in documentation practices.

References

Bosch, J. (2004). Software architecture: The next step, In *European workshop on software architecture*. Springer.

De Boer, R. C., Farenhorst, R., Lago, P., Van Vliet, H., Clerc, V., & Jansen, A. (2007). Architectural knowledge: Getting to the core, In *International conference on the quality of software architectures*. Springer.

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

ISO. (2011). Iec/ieee systems and software engineering: Architecture description. *ISO/IEC/IEEE 42010: 2011 (E)(Revision of ISO/IEC 42010: 2007 and IEEE Std 1471-2000)*.

Kruchten, P. (2004). An ontology of architectural design decisions in software intensive systems, In *2nd groningen workshop on software variability*. Citeseer.

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

Kruchten, P., Capilla, R., & Dueñas, J. C. (2009). The decision view's role in software architecture practice. *IEEE software*, *26*(2), 36–42.

Liang, P., Jansen, A., & Avgeriou, P. (2009). *Knowledge architect: A tool suite for managing software architecture knowledge*. University of Groningen, Johann Bernoulli Institute for Mathematics; Computer Science.

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, *17*(4), 40–52.

Tang, A., Avgeriou, P., Jansen, A., Capilla, R., & Babar, M. A. (2010). A comparative study of architecture knowledge management tools. *Journal of Systems and Software*, *83*(3), 352–370.

Tyree, J., & Akerman, A. (2005). Architecture decisions: Demystifying architecture. *IEEE software*, *22*(2), 19–27.

van de Weerd, I., & Brinkkemper, S. (2009). Meta-modeling for situational analysis and design methods, In *Handbook of research on modern systems analysis and design technologies and applications*. IGI Global.

Van Gurp, J., & Bosch, J. (2002). Design erosion: Problems and causes. *Journal of systems and software*, *61*(2), 105–119.

Van Heesch, U., Avgeriou, P., & Hilliard, R. (2012a). A documentation framework for architecture decisions. *Journal of Systems and Software*, *85*(4), 795–820.

Van Heesch, U., Avgeriou, P., & Hilliard, R. (2012b). Forces on architecture decisions-a viewpoint, In *2012 joint working ieee/ifip conference on software architecture and european conference on software architecture*. IEEE.

van Heesch, U., Jansen, A., Pei-Breivold, H., Avgeriou, P., & Manteuffel, C. (2017). Platform design space exploration using architecture decision viewpoints–a longitudinal study. *Journal of Systems and Software*, *124*, 56–81.