# DecDoc: A Tool for Documenting Design Decisions Collaboratively and Incrementally

Tom-Michael Hesse*, Arthur Kuehlwein*, and Tobias Roehm†
*Heidelberg University, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany
{hesse,kuehlwein}@informatik.uni-heidelberg.de
†CQSE GmbH, Lichtenbergstrasse 8, 85748 Garching b. München, Germany, roehm@csqe.eu

***Abstract*—The outcome and quality of design decisions highly depend on the knowledge reflected during decision-making. Typically, making design decisions is not one singular action. Instead, developers discuss and cooperate during requirements engineering, design and implementation of a system to make and adapt design decisions. This decision-making process is influenced by different decision-making strategies, personal experiences, and biases. In consequence, decision-related knowledge emerges incrementally over time in an incomplete and heterogeneous way. This hinders the documentation of such knowledge in practice. First, most documentation tools capture decision-related knowledge within one particular development activity. However, they do not focus on the collaborative and shared documentation during multiple activities. Second, static documentation templates and formal rules are not suitable for capturing incomplete knowledge, as additional documentation effort is imposed for developers. Thus, text templates are not used or filled with generic contents. As a result, decision-related knowledge remains implicit and is not available to guide future decision-making. To address these issues, we have created the tool DecDoc based on our incremental documentation model. The tool enables developers to capture decision-related knowledge and collaborate on a comprehensive documentation of design decisions with relations to artifacts, such as requirement specifications, design diagrams, and code. This helps to improve the decision-making process for design decisions, as it helps to make explicit and reflect related knowledge during the process. In this paper, we present DecDoc with regard to requirements from the decision-making process. Then, we describe its application on design decisions in example projects. Finally, we discuss our insights from using the tool and highlight open challenges.**

***Index Terms*—Decision documentation, design decisions, decision knowledge, design decision-making, decision capturing, design documentation, knowledge representation**

## I. Introduction

Developers make many design decisions, which turn out to have a critical importance for the project's success during multiple development activities. A broadly acknowledged example are decisions on how to realize architecturally significant requirements during the system's design and implementation [1]. In order to make decisions, developers need to solve decision problems, which consist of a set of alternatives and criteria to compare them [2]. A comparison of alternatives typically requires expert knowledge, personal experiences and the context of the decision [3]. Thus, large and complex amounts of decision-related knowledge are created, reflected, and evolved during the decision-making process [4], [5]. We refer to this knowledge as *decision knowledge*.

Decision-making processes are often performed in collaboration of multiple developers, when the decisions affects different development activities. Then, the decision is made incrementally. Documenting the emerging decision knowledge is crucial for enabling developers to communicate and reflect these decision-making processes and their resulting decisions. Often, communication or reflection is necessary, as development teams change, or the system's design needs to be adapted in follow-up decisions. Then, documented decision knowledge helps to guide and improve future decision-making, for instance by identifying patterns of strengths and weaknesses in current decisions [6]. However, decision knowledge erodes quickly and might be even lost completely [4], if it is not documented within or after decision-making processes. A study of Tang et al. [7] found that designers miss specialized methods and tool support to document decision knowledge. Thus, our *overall goal* is to create *a documentation tool for decision knowledge, which is aligned with a collaborative and incremental decision-making process*. To integrate decision documentation tools closely with decision-making processes, two requirements have to be addressed. First, the tool should enable developers to collaboratively work on a shared decision documentation. Second, it should be possible to document decision knowledge incrementally in a fine-grained way.

These two requirements are not sufficiently fulfilled by current documentation tools, as described in a detailed comparison in Section II-C. In contrast, our tool DecDoc provides features to enable both collaborative and incremental documentation of design decisions. In this paper, we describe the requirements for decision documentation during collaborative and incremental decision-making. Then, we demonstrate our tool with decision examples from two different projects. In addition, we discuss open challenges that we have observed when investigating related work and applying our tool.

The remainder of this paper is structured as follows. In Section II, we describe the requirements addressed by our approach, and evaluate related tools. Section III provides an overview of our documentation tool DecDoc. Then, we describe the usage of DecDoc for realistic design decisions in Section IV. Afterwards, we describe and discuss open challenges for DecDoc in particular and decision documentation in general in Section V. Finally, we conclude our insights and describe ideas for future work in Section VI.

## II. Requirements and Related Work

In this section, we introduce a *running example* to illustrate our problem and solution description. Then, we introduce the problem of collaborative and incremental decision-making processes with the *resulting requirements*. In addition, we present an overview of *related work* and investigate, to which degree related tools fulfill our requirements.

### A. Running Example

In the following sections, we use example design decisions of the CoCoME project to explain and illustrate our tool. CoCoME represents a trading system for sales management in supermarket enterprises. Details are described in [8]. In particular, the system covers the sales processes within single stores as well as the enterprise-wide inventory management for product orders to suppliers. In this paper, we assume that the *system architect Alice* makes the architectural decision to migrate major parts of CoCoME to the cloud (referred to as **1a**). This decision is beneficial for the scalability and cost-effectiveness of the system. However, it also impacts the security requirements and implementation of the application. For instance, *requirements engineer Bob* is concerned whether the security of payment data can be ensured in distributed cloud storage (**1b**). In addition, *developer Carol* discovers during code adaptions that the architectural change is constrained by the system connectivity of the suppliers (**1c**). Finally, the original design decision is reconsidered by Alice and causes follow-up decisions (**2**).

### B. Requirements for Documentation

The collaborative and incremental decision-making process in our running example is depicted in Figure 1.

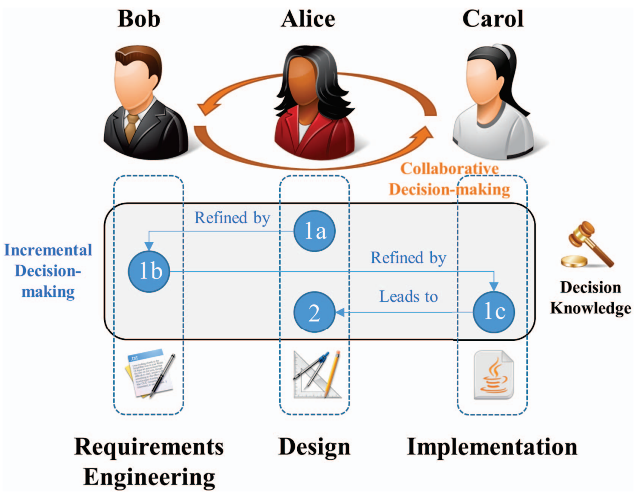


Fig. 1. A Collaborative and Incremental Decision-making Process

The example highlights that multiple developers make and refine decisions in a group decision-making situation [9]. This situation complicates the documentation of decision knowledge. First, multiple developers contribute to the decision, so that decision knowledge needs to be captured during different development activities, and in relation to the affected artifacts. For instance, developers collaborate during requirements engineering, design and implementation to make and enforce design decisions on architecturally significant requirements [10]. Due to the nature of such requirements, the related decisions have a wide impact on the system. However, these decisions evolve when the addressed requirements change over time [1]. Therefore, it is important to bundle all knowledge related to one decision to make it comprehensible for developers [11]. This is even more important, as developers may be dispersed in different locations [9]. In consequence, developers should be enabled to *work collaboratively on the same documentation of decision knowledge during different development activities* (requirement **R1**).

Second, not all decision knowledge is available at once, as the collaborative decision-making process consists of multiple actions and discussions. For instance, former design decisions are adapted and refined in follow-up development iterations [12]. In addition, different developers may be influenced by different decision-making strategies, such as rational or naturalistic decision-making (abbreviated as RDM and NDM) [3]. Then, also the related knowledge varies. Alternatives and assessments are important components of RDM [13], as the decision solution is determined by *choice*. In contrast, claims and scenarios are typical for a *match*-based identification of solutions in NDM [14]. Thus, the complete set of decision knowledge is not available all at once. Instead, it grows over time, and is impacted by the decision-making strategies applied by each developer. In consequence, it should be possible to *document decision knowledge incrementally in a fine-grained way* (requirement **R2**).

### C. Related Work

We have investigated to which degree tools from related work fulfill our requirements. Relevant tools were identified based on comparative studies by Ali Babar et al. [15], Tang et al. [11], and our own study [16]. In addition, we added two recent tools of Cleland-Huang et al. [17] and Manteuffel et al. [18]. To cope with the described problems, decision documentation approaches should sufficiently fulfill both requirements R1 and R2.

For R1, we examined during which development activities the tool can be used to document decisions. In addition, we evaluated how different documentation aspects were integrated. This can be realized by sharing the documentation, importing knowledge, providing links, or not at all. We consider R1 to be sufficiently fulfilled if all three development activities are supported for decision documentation and working on shared documentation is enabled. The results are summarized in Table I. Several tools do support decision documentation for two development activities, mostly for requirements engineering and design. LISA implicitly supports all three development activities, as an integrated representation for requirements and design decisions is provided. However, this limits the support for complex specifications of requirements, such as

TABLE I
TOOL COMPARISON REGARDING COLLABORATIVE DOCUMENTATION (R1)

| Tool | Supported Activities | Integration of Documentation |
|---|---|---|
| ADDSS [19] | RE, D | Links, Shared Documentation |
| ADkwik [20] | D | None |
| Archie [17] | RE, D, I | Links, Knowledge Import |
| Architecture Warehouse [21] | D | Links, Shared Documentation |
| Archium [4] | RE, D | Links |
| AREL [22] | RE, D | Links |
| CoCoADvISE [23] | D | Shared Documentation |
| COMANCHE [24] | D, I | Links, Shared Documentation |
| Enterprise Architect add-in [18] | RE, D | Links, Shared Documentation |
| Knowledge Architect [25] | RE, D | Links, Knowledge Import, Shared Documentation |
| LISA [26] | (RE), D, I | Links, Shared Documentation |
| PAKME [27] | RE, D | Links |
| SEURAT [28] | RE, I | Links, Knowledge Import, Shared Documentation |

RE = Requirements Engineering, D = Design, I = Implementation

use cases. Knowledge Architect and SEURAT provide the most options for the integration of different documentation aspects. In contrast, Archie also provides explicit support for implementation by integrating architectural decisions and codes files. However, it does not cover a shared documentation of decisions. Also, the focus of Archie is more on traceability between development artifacts and not on decision documentation itself.

TABLE II
TOOL COMPARISON REGARDING INCREMENTAL DOCUMENTATION (R2)

| Tool | Decision Structure | Iteration Support | Refinement Support | Strategy Support |
|---|---|---|---|---|
| ADDSS [19] | Fixed | Yes | Refinement entities, sub-types | Choice |
| ADkwik [20] | Fixed | Yes | Refinement relations | Choice |
| Archie [17] | Monolithic | No | Sub-types | Choice |
| Architecture Warehouse [21] | Fixed | Yes | No refinement | Choice |
| Archium [4] | Monolithic | No | No refinement | Choice |
| AREL [22] | Monolithic | Yes | Refinement entities | Choice |
| CoCoADvISE [23] | Fixed | Yes | No refinement | Choice, match |
| COMANCHE [24] | Fixed | Yes | No refinement | Choice |
| Enterprise Architect add-in [18] | Fixed | Yes | No refinement | Choice |
| Knowledge Architect [25] | Monolithic | No | Refinement entities | Choice |
| LISA [26] | Monolithic | Yes | Refinement relations | Choice |
| PAKME [27] | Monolithic | No | No refinement | Match |
| SEURAT [28] | Flexible | Yes | Refinement entities and relations | Choice |

For R2, we examined documentation structures prescribed by the tool and the support of iterations within these structures. Documentation structures can be monolithic like static text templates, contain fixed relations, or offer flexible compositions and aggregations. We also investigated whether the refinement of decision knowledge was supported by fine-grained documentation entities. Moreover, we were interested in any implicit or explicit preference for decision-making strategies according to how solutions were documented within a tool. This could be either "choice" for RDM or "match" for NDM according to the strategy's mechanism for solving decision problems [3]. We consider R2 to be sufficiently addressed if tools provide a flexible documentation structure with support for iterations, refinements, and both choice and match of solutions. The results are summarized in Table II. SEURAT is the only tool, which provides a flexible structure for decision documentation. SEURAT provides support for documentation iterations with entities and relations for knowledge refinement. However, like most other tools it assumes a RDM-based solution selection by choice. In contrast, CoCoADvISE and PAKME are the only tools, which explicitly support the documentation of matching solutions within architectural decisions. All other tools rely either on a monolithic or a fixed structure. Further existing approaches provide flexible structures with support for refinements and iterations, such as pattern-based decision models [29] and decisions as reusable design assets [30]. However, no tools could be found, which implement these approaches.

Overall, it should be noted that Archie and SEURAT address the most aspects of both requirements in comparison to all investigated tools. Some tools address one requirement sufficiently, such as LISA for R1. However, we did not uncover a tool, which fulfills both requirements.

## III. SOLUTION APPROACH OF DECDOC

In this section, we introduce our model for *decision documentation*, which is implemented in our solution approach DecDoc. Then, we briefly describe the *technical foundations* of DecDoc and present its documentation *features* in detail.

### A. Decision Documentation

Several models currently exist to document decision knowledge. In [16], we have presented an overview about these documentation models for design decisions. However, no model integrates decision knowledge documentation for requirements engineering, design and implementation. In addition, most models mainly focus on capturing decisions in monolithic knowledge entities or static text templates without support for incremental documentation.

Due to these shortcomings, we decided to develop our own documentation model for DecDoc, as presented in [31]. The entire model is depicted in Figure 2. It provides a representation for general development artifacts as *Knowledge Elements* with specialized *Decision Knowledge Elements* to

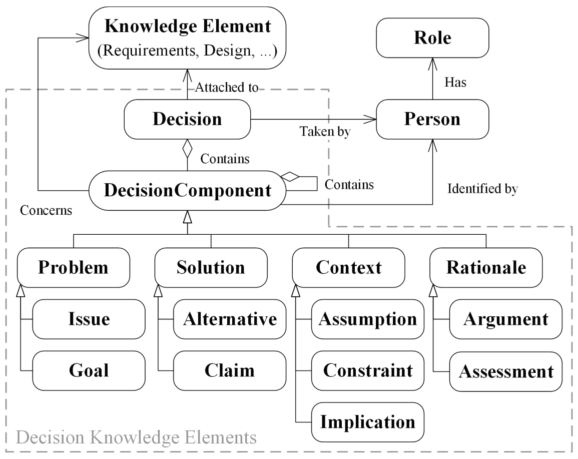


Fig. 2. Decision Documentation Model (Source: [31])

model particular aspects of decision knowledge. The basic element is *Decision*, which contains all related decision knowledge as *DecisionComponents*. Decision knowledge elements can be added incrementally over time by different *Persons*. Decisions and DecisionComponents can be linked to other knowledge elements, for instance to requirements, design artifacts or code files. Different kinds of DecisionComponents are distinguished to describe the decision's *Problem* and *Solution*, its *Context* and *Rationale*. *Issues* or *Goals* can be used to document details on open questions for a decision, whereas *Alternatives* and *Claims* represent options to solve the decision problem. Context information may consist of *Assumptions* influencing the decision, *Constraints* restricting the decision, or *Implications* resulting from different alternatives. Reasons for or against decision knowledge elements can be expressed as *Arguments*, whereas the evaluation of criteria is documented as *Assessment*.

### B. Technical Foundations

Our documentation tool DecDoc is an extension to the model-based knowledge management tool *UNICASE* [33] and based on the *Eclipse IDE*, as depicted in Figure 3. The tool is available via an update site [32]. The knowledge management tool UNICASE provides an integrated model for system and project knowledge with generic support for collaborative editing of various elements, such as use cases, UML diagrams, or code revisions. Both UNICASE and DecDoc employ the

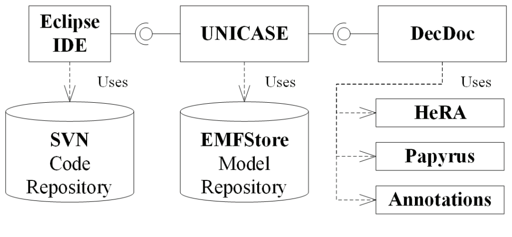


Fig. 3. Architecture of DecDoc

versioned model repository *EMFStore* [34] to persist the knowledge models and their instances. In addition, DecDoc imports knowledge from results produced by the *HeRA* plugin for Eclipse [35], which heuristically analyzes use cases for security concerns. DecDoc also integrates with the *Papyrus* UML editor [36] for Eclipse to provide relations between decisions and UML entities. Moreover, Eclipse *markers and code annotations* are used to represent decision knowledge within source code. Therefore, DecDoc synchronizes the decision knowledge stored in the EMFStore with annotations stored in revisions of the code repository SVN.

### C. Features of DecDoc

DecDoc provides several features to address the requirements R1 and R2. An overview of these features is depicted in Figure 4.

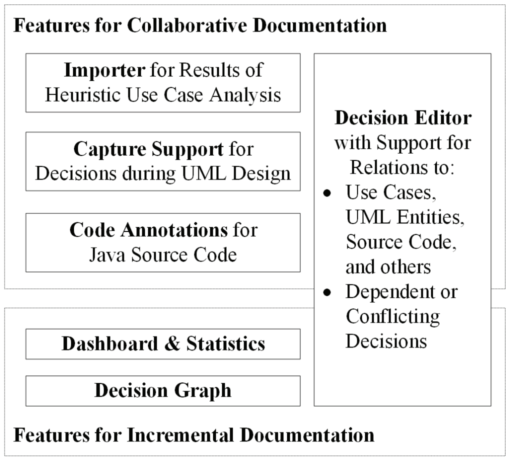
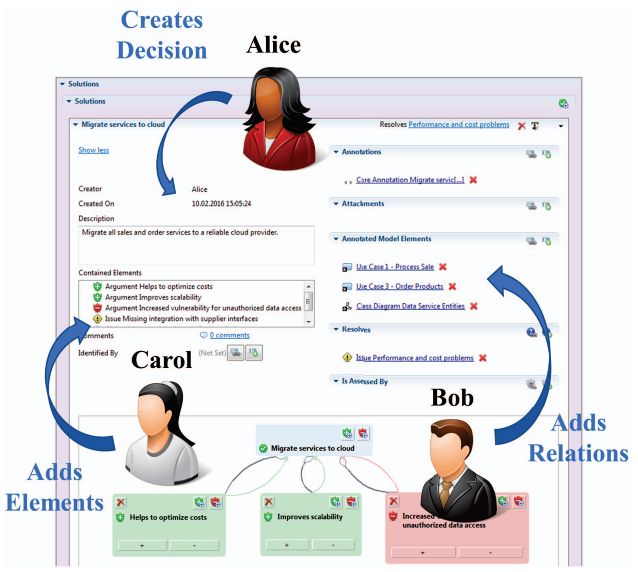

Features for Collaborative Documentation
Importer for Results of Heuristic Use Case Analysis
Capture Support for Decisions during UML Design
Code Annotations for Java Source Code
Decision Editor with Support for Relations to:
Use Cases, UML Entities, Source Code, and others
Dependent or Conflicting Decisions
Dashboard & Statistics
Decision Graph
Features for Incremental Documentation

Fig. 4. Features provided by DecDoc

General editing support for decisions is provided by the *Decision Editor*. Within this editor, developers can create and edit any DecisionComponents collaboratively, as depicted in Figure 5. Regarding our example, Alice initially documents her architectural decision on the cloud migration with a description and basic attributes, such as the current decision-making progress or implementation status. Bob and Carol may extend the decision concurrently with further information. For instance, Bob links the decision to any dependent security-related decisions for the affected use cases. Carol adds further arguments and context information, as she proceeds with the decision implementation. Thus, a *shared documentation* is enabled. The developers can work at different locations, as the decision knowledge is versioned and exchanged via the EMFStore.

*Features Supporting the Collaborative Documentation:* DecDoc supports the collaborative work on the shared documentation during design, requirements engineering, and implementation. First, architects and designers can *capture design decisions when editing UML diagrams*. This is depicted in Figure 6. In our example, Alice captures her decision on


Creates Decision
Alice
Carol
Bob
Adds Elements
Adds Relations
Solutions
Solutions
Migrate services to cloud
Resolves Performance and cost problems
Show less
Annotations
Core Annotation Migrate servic[...]
Attachments
Annotated Model Elements
Use Case 1 - Process Sale
Use Case 3 - Order Products
Class Diagram Data Service Entities
Resolves
Issue Performance and cost problems
Is Assessed By
Migrate services to cloud
Helps to optimize costs
Improves scalability
Increased unauthorized data access

Creator Alice
Created On 10.02.2016 15:05:24
Description
Migrate all sales and order services to a reliable cloud provider.
Contained Elements
Argument Helps to optimize costs
Argument Improves scalability
Argument Increased vulnerability for unauthorized data access
Issue Missing integration with supplier interfaces
Comments 0 comments
Identified By (Not Set)

Fig. 5. Decision Editor with Decision Knowledge from Different Activities

how to adapt the architecture of CoCoME to support a cloud environment (1a), as she adapts the UML diagram containing stored products and their orders. DecDoc allows her to create a decision directly within the diagram, so that no context switch is necessary. Moreover, the decision is linked to the selected UML classes. This enables the developers to trace the decision of Alice to all affected UML entities.

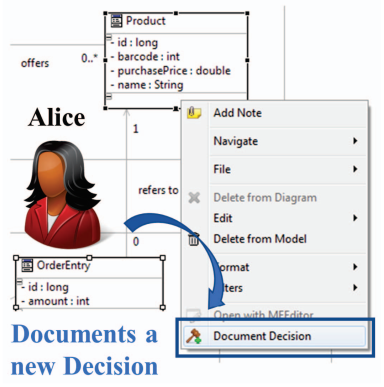
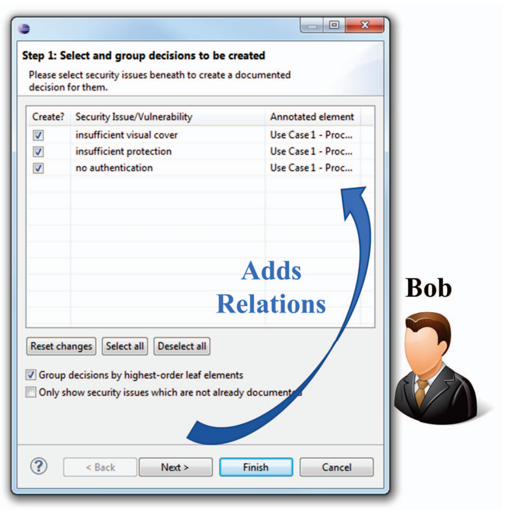


Fig. 6. Documentation of Decisions in UML Diagrams during Design

Second, DecDoc provides a semi-automatic *importer for decision knowledge from heuristic security analyses of use cases*. Therefore, the Eclipse HeRA plugin is used, as described in detail in [35]. This demonstrates the ability of DecDoc to incorporate knowledge from external knowledge sources. As the developers continue their decision-making process for the CoCoME cloud migration, requirements engineer Bob reviews and updates all affected use cases. Then he checks with HeRA whether the changes could impact system security. This is depicted in Figure 7. For instance, Bob discovers potential data protection issues for payment information after preparing the sales use case for the cloud setup. Bob discusses these issues with Alice and Carol, and they decide to refine their migration decision (1a to 1b). The documentation of this refinement is based on generated candidates for decision knowledge elements from the HeRA results.



**Step 1: Select and group decisions to be created**
Please select security issues beneath to create a documented decision for them.

| Create? | Security Issue/Vulnerability | Annotated element |
|---|---|---|
| ✓ | insufficient visual cover | Use Case 1 - Proc... |
| ✓ | insufficient protection | Use Case 1 - Proc... |
| ✓ | no authentication | Use Case 1 - Proc... |

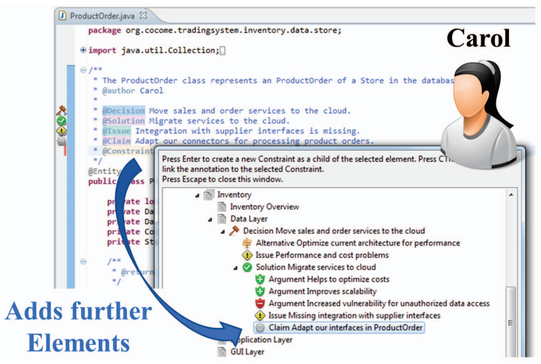
[✓] Group decisions by highest-order leaf elements
[ ] Only show security issues which are not already documented

Fig. 7. Extend Documentation by Incorporating Heuristic Analysis Results during Requirements Engineering

Third, DecDoc enables developers to document their design decisions within the source code using *code annotations*. Therefore, all decision knowledge elements are mapped to corresponding annotations, such as `@Issue` for issues, or `@Claim` for claims. Annotations can be used to either document new decision knowledge or link existing decision knowledge elements with code. We have presented the annotations in detail in [37]. For the CoCoME cloud migration, developer Carol starts to adapt the implementation of the `ProductOrder` class and other classes implementing sales and order processes. This is depicted in Figure 8. She recognizes a dependency of the order process to the interfaces of the suppliers (1b to 1c). Thus, she creates a new constraint for the decision on cloud migration with a code annotation.

Overall, DecDoc *integrates documented decision knowledge* from requirements engineering, design, and implementation with links to related artifacts. Moreover, DecDoc can be extended by using defined Eclipse extension points. For instance, further external knowledge sources can be integrated via Eclipse plugins, or new code annotations can be added. All documented knowledge is bundled in a shared documentation, which is accessible via the decision editor. Thus, DecDoc fulfills requirement R1.

*Features Supporting the Incremental Documentation:* The documentation model for DecDoc supports the *incremental aggregation* of DecisionComponents over time, as the decision-making process proceeds. The model does not prescribe a static structure for decisions, and, therefore, is *flexible*. In addition, different kinds of DecisionComponents are distinguished,



ProductOrder.java

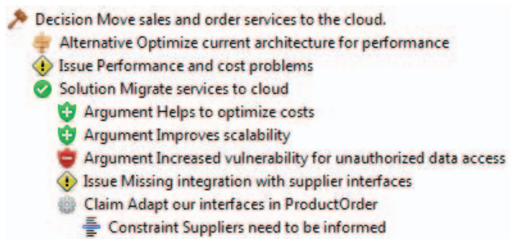
```
package org.cocome.tradingsystem.inventory.data.store;

import java.util.Collection;

/**
 * The ProductOrder class represents an ProductOrder of a Store in the databas
 * @author Carol
 *
 * @Decision Move sales and order services to the cloud.
 * @Solution Migrate services to the cloud.
 * @Issue Integration with supplier interfaces is missing.
 * @Claim Adapt our connectors for processing product orders.
 * @Constraint
 */
@Entity
public  ass P

    pr ate lo
    pr ate Da
    pr ate Da
    pri ate Co
    pri ate St

    /**
     * @re urn
     */
```

Press Enter to create a new Constraint as a child of the selected element. Press C
link the annotation to the selected Constraint.
Press Escape to close this window.

- Inventory
  - Inventory Overview
  - Data Layer
    - Decision Move sales and order services to the cloud
      - Alternative Optimize current architecture for performance
      - Issue Performance and cost problems
      - Solution Migrate services to cloud
        - Argument Helps to optimize costs
        - Argument Improves scalability
        - Argument Increased vulnerability for unauthorized data access
        - Issue Missing integration with supplier interfaces
        - Claim Adapt our interfaces in ProductOrder
  - plication Layer
  - GUI Layer

Fig. 8. Extend Documentation with Code Annotations during Implementation

such as Problem, Solution, Rationale, or Context. This enables developers to *refine given knowledge* by adding further fine-grained decision knowledge elements. These features may lead to complex and nested knowledge structures, as depicted in Figure 9. The example structure shows our cloud migration decision (1a, 1b and 1c) with the knowledge created by Alice, Bob and Carol during their respective development activities. Within this tree structure of the knowledge elements, the different levels represent the depth of the contains-relation. For instance, the depicted arguments are contained within the solution "Migrate services to cloud". Whereas Alice and Bob mostly documented rational decision-making elements, Carol also used elements related to NDM. For instance, she created a Claim to describe the adaptation of the CoCoME interfaces for a product order during implementation. Thus, our tool allows for a *mixed documentation of both NDM and RDM* within one decision. This addresses the findings of Zannier et al. [3] that developers tend to mix up both decision-making strategies.



- Decision Move sales and order services to the cloud.
  - Alternative Optimize current architecture for performance
  - Issue Performance and cost problems
  - Solution Migrate services to cloud
    - Argument Helps to optimize costs
    - Argument Improves scalability
    - Argument Increased vulnerability for unauthorized data access
    - Issue Missing integration with supplier interfaces
    - Claim Adapt our interfaces in ProductOrder
      - Constraint Suppliers need to be informed

Fig. 9. Incremental Decision Knowledge Structure in DecDoc

To enable developers to keep track with more complex documentation structures, DecDoc provides a *dashboard with statistics* for each documented decision. It calculates the number of all contained decision elements grouped by their type, and shows all relations of the decision to other knowledge elements. To visualize the emerged knowledge structures more in detail, DecDoc offers a graph visualization of relations between decisions, and for decisions with their contained elements.

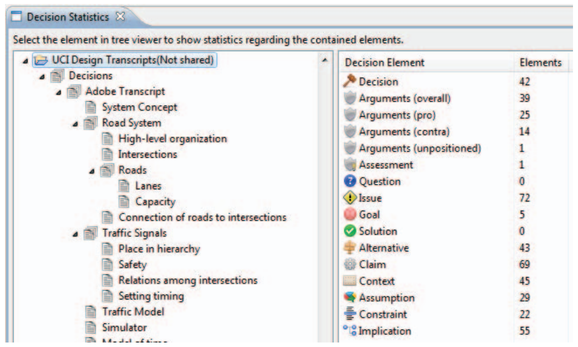
In summary, DecDoc provides a flexible knowledge structure with support for RDM and NDM knowledge elements. Developers can refine documented decisions by either adding new fine-grained knowledge elements of different kinds or by editing existing elements. DecDoc enables developers to explore the documented knowledge using the dashboard or the graph visualization. Thus, DecDoc fulfills requirement R2.

## IV. Evaluation of DecDoc

We applied DecDoc on more complex and realistic data to evaluate its feasibility for documenting design decisions. Therefore, we have investigated transcripts of two design sessions held by professional software designers from Adobe and Amberpoint. The transcripts were initially distributed as material for the international workshop "Studying Professional Software Design" in 2010 [38]. In each design session, two designers were asked to create the architecture for a traffic simulation system according to basic requirements. The workshop organizers requested the designers to think aloud and to state all thoughts explicitly. Both sessions were video-recorded and transcribed afterwards. During the discussions, many design decisions were made collaboratively, as both designers exchanged thoughts and arguments on the requirements, design and implementation of the system. In addition, the designers revised and adapted their decisions during the sessions. Thus, also the incremental aggregation of decision knowledge can be observed.

We analyzed the discussion transcripts for contained decision knowledge and extracted it according to our documentation model. Thus, we performed a retrospective analysis of decision-making processes. Details can be found in [39]. The extracted knowledge was documented with DecDoc. A summary from DecDoc statistics is shown in Figure 10. In total, we created 42 decisions with 380 decision knowledge elements for both transcripts. In particular, we identified 69 claims within all decisions. This indicates that all designers have applied not only RDM, but also NDM in their decisions. In addition, it confirms that providing documentation elements related to NDM is useful and important.

We also observed that complex knowledge structures emerged during the design discussions. An example is the decision on where to place traffic lights within the architecture of the simulation system. For this decision, four different levels



Decision Statistics

Select the element in tree viewer to show statistics regarding the contained elements.

- UCI Design Transcripts(Not shared)
  - Decisions
    - Adobe Transcript
      - System Concept
      - Road System
        - High-level organization
        - Intersections
        - Roads
          - Lanes
          - Capacity
        - Connection of roads to intersections
      - Traffic Signals
        - Place in hierarchy
        - Safety
        - Relations among intersections
        - Setting timing
      - Traffic Model
      - Simulator

| Decision Element | Elements |
|---|---|
| Decision | 42 |
| Arguments (overall) | 39 |
| Arguments (pro) | 25 |
| Arguments (contra) | 14 |
| Arguments (unpositioned) | 1 |
| Assessment | 1 |
| Question | 0 |
| Issue | 72 |
| Goal | 5 |
| Solution | 0 |
| Alternative | 43 |
| Claim | 69 |
| Context | 45 |
| Assumption | 29 |
| Constraint | 22 |
| Implication | 55 |

Fig. 10. Statistics for Design Decisions on Traffic Lights

of contains-relations between decision knowledge elements were found. For instance, the designers used implications and arguments to refine a claim. This confirms that it is useful and important to capture design decisions incrementally when the decision-making process of the designers proceeds during their discussions. Altogether, we showed that it is feasible to document complex decision knowledge in DecDoc from collaborative and incremental decision-making processes.

## V. Open Challenges

By applying DecDoc for realistic decision data, we have demonstrated that the tool is capable of capturing complex knowledge structures, which emerged incrementally during the collaborative decision-making process of multiple developers.

However, our flexible documentation approach allows for incompleteness of the documented knowledge. This may complicate the reflection and further analysis of the documentation. Thus, it is important to provide further imports for decision knowledge from external sources. For instance, analyses of requirements and design specifications in Word documents (cf. [25]) can provide further semi-automatic input for decision knowledge from external artifacts. This lowers the manual documentation effort for developers. In addition, a mixed documentation according to different decision-making strategies and different abstraction levels of knowledge elements might decrease the comprehensibility of the documentation. Developers need to identify and discuss such issues. This is supported by the dashboard and visualizations. Then, the developers may add the results as new knowledge elements to optimize the documentation. However, more sophisticated support for reviews needs to be provided. Also, we investigated only the retrospective documentation of decision-making processes. Further studies are required to evaluate the feasibility of DecDoc for documenting decisions during discussions for decision-making. Moreover, the incremental and collaborative documentation is not the only concern of documentation-related quality issues of decision-making. The following list presents further challenges for documentation during decision-making with regard to our tool and related work:

*Consistency between Documentation and Implementation of Decisions:* Documentation tools could support consistency checks between documented knowledge and decision implementation (cf. [17], [40]). This would help to ensure that all developers are aware of the decision and implement it, unless it is challenged explicitly. However, such consistency checks are difficult to realize based on documentation. Typically, they require formal verification methods for both decision documentation and implementation artifacts, such as design diagrams, or source code. Thus, investigating and realizing such consistency checks requires dedicated research projects.

*Focus on Important Decisions:* It is not realistic that developers document all design decisions explicitly in practice. Major reasons are the required documentation effort and differences in creative and reflective thinking during decision-making [41]. Thus, our documentation tool and many related tools focus on design decisions with importance for the success of the development project, such as architectural design decisions (cf. [4], [19], [20]). However, it is an open question how to determine the importance of decisions early during decision-making. For instance, it is difficult to identify decisions on architecturally significant requirements [1]. Our tool allows for documentation at different levels of granularity and with different kinds of knowledge entities. This depends on the importance of a decision as perceived by developers. Then, reviewers are required during the decision-making process, who check that the importance of decisions was estimated correctly [41].

*Maintenance of Documentation:* Many existing approaches address the evolution of documented decisions (cf. [5], [42]). Our tool helps to maintain documented decision knowledge, as many developers can collaboratively maintain this documentation. This alleviates the recognition and adaption of outdated or incorrect documentation. In addition, consistency checks between the documentation and implementation of decisions would be helpful to uncover inconsistencies. However, the principal problem remains that some decision knowledge becomes outdated or even wrong, as the system under development and its design decisions evolve. This cannot be addressed automatically by decision documentation tools. Instead, developers are required to explicitly maintain their documentation and update it, if necessary.

## VI. Conclusion and Future Work

In this paper, we have presented our documentation tool DecDoc. The tool addresses two requirements: Developers are enabled to document design decisions *collaboratively* during different development activities and *incrementally* with fine-grained documentation elements. We investigated current decision documentation tools and found that no single tool fulfilled both requirements sufficiently. In contrast, our tool enables developers to work on a shared documentation of design decisions during requirements engineering, design and implementation. DecDoc provides an incremental and fine-grained knowledge model for decision documentation. Links to other development artifacts, such as use cases, UML diagrams, or code files, are supported. In addition, knowledge from external sources, like heuristic analysis results for use case descriptions, can be imported. We have applied our tool on realistic data of design decisions from professional software designers. Therefore, we documented 42 decisions with 380 decision knowledge elements to show that it is feasible to document complex knowledge structures in DecDoc.

We plan future work in two directions. First, we aim to evaluate our tool in further case studies with professional designers to investigate how DecDoc can be employed directly by designers during decision-making. Also, it should be evaluated for which design decisions the usage of which decision knowledge elements is beneficial to maintain a comprehensive, but cost-efficient documentation. Second, we want to further improve DecDoc. For instance, support for other code repositories and more imports for decision knowledge should be implemented.

## Acknowledgment

This work was partially supported by the German Research Foundation under the Priority Programme SPP1593: Design For Future — Managed Software Evolution. Results described in Section IV are based upon videos and transcripts initially distributed for the 2010 international workshop "Studying Professional Software Design", as partially supported by NSF grant CCF-0845840. We thank Barbara Paech for very helpful discussions and feedback.

## References

[1] L. Chen, M. Ali Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *Software*, vol. 30, no. 2, pp. 38–45, 2013.

[2] T. Ngo and G. Ruhe, "Decision support in requirements engineering," in *Engineering and Managing Software Requirements*. Springer, 2005, pp. 267–286.

[3] C. Zannier, M. Chiasson, and F. Maurer, "A model of design decision making based on empirical results of interviews with software designers," *Information and Software Technology*, vol. 49, no. 6, pp. 637–653, 2007.

[4] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. IEEE, 2005, pp. 109–120.

[5] R. Capilla, F. Nava, and J. C. Duenas, "Modeling and documenting the evolution of architectural design decisions," in *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07)*. IEEE, 2007.

[6] J. A. Maule, "Can computers help overcome limitations in human decision making?" *Int. Journal of Human-Computer Interaction*, vol. 26, no. 2-3, pp. 108–119, 2010.

[7] A. Tang, M. Ali Babar, I. Gorton, and J. Han, "A survey of architecture design rationale," *Journal of Systems and Software*, vol. 79, no. 12, pp. 1792–1804, 2006.

[8] S. Herold, H. Klus, Y. Welsch, C. Deiters *et al.*, "Cocome - the common component modeling example," in *The Common Component Modeling Example*, A. Rausch, R. Reussner, R. Mirandola, and F. Plášil, Eds. Springer, 2008, pp. 16–53.

[9] V. Smrithi Rekha and H. Muccini, "A study on group decision-making in software architecture," in *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*. IEEE, 2014, pp. 185–194.

[10] B. Nuseibeh, "Weaving together requirements and architectures," *IEEE Computer*, vol. 34, no. 3, pp. 115–119, 2001.

[11] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar, "A comparative study of architecture knowledge management tools," *Journal of Systems and Software*, vol. 83, no. 3, pp. 352–370, 2010.

[12] A. J. Ko and P. K. Chilana, "Design, discussion, and dissent in open bug reports," in *Proceedings of the 2011 iConference*, 2011, pp. 106–113.

[13] R. Lipshitz, G. Klein, J. Orasanu, and E. Salas, "Taking stock of naturalistic decision making," *Journal of Behavioral Decision Making*, vol. 14, no. 5, pp. 331–352, 2001.

[14] J. M. Carroll and M. B. Rosson, "Getting around the task-artifact cycle: how to make claims and design by scenario," *ACM Transactions on Information Systems*, vol. 10, no. 2, pp. 181–212, 1992.

[15] M. Ali Babar, R. C. de Boer, T. Dingsoyr, and R. Farenhorst, "Architectural knowlege management strategies: Approaches in research and industry," in *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07)*. IEEE, 2007, pp. 35–41.

[16] B. Paech, A. Delater, and T.-M. Hesse, "Integrating project and system knowledge management," in *Software Project Management in a Changing World*, G. Ruhe and C. Wohlin, Eds. Springer, 2014, pp. 157–192.

[17] J. Cleland-Huang, M. Mirakhorli, A. Czauderna, and M. Wieloch, "Decision-centric traceability of architectural concerns," in *7th Int. Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'13)*. IEEE, 2013, pp. 5–11.

[18] C. Manteuffel, D. Tofan, H. Koziolek, T. Goldschmidt, and P. Avgeriou, "Industrial implementation of a documentation framework for architectural decisions," in *11th Working IEEE/IFIP Conference on Software Architecture (WICSA'14)*, 2014, pp. 225–234.

[19] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, "A web-based tool for managing architectural design decisions," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 5, pp. 1–8, 2006.

[20] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, "Managing architectural decision models with dependency relations, integrity constraints, and production rules," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1249–1267, 2009.

[21] M. Nowak and C. Pautasso, "Team situational awareness and architectural decision making with the software architecture warehouse," in *Software Architecture: 7th European Conference (ECSA'13)*. Springer, 2013, pp. 146–161.

[22] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *Journal of Systems and Software*, vol. 80, no. 6, pp. 918–934, 2007.

[23] P. Gaubatz, I. Lytra, and U. Zdun, "Automatic enforcement of constraints in real-time collaborative architectural decision making," *Journal of Systems and Software*, vol. 103, pp. 128–149, 2015.

[24] G. Canfora, G. Casazza, and A. De Lucia, "A design rationale based environment for cooperative maintenance," *Int. Journal of Software Engineering and Knowledge Engineering*, vol. 10, no. 5, pp. 627–645, 2000.

[25] A. Jansen, P. Avgeriou, and J. S. van der Ven, "Enriching software architecture documentation," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1232–1248, 2009.

[26] G. Buchgeher and R. Weinreich, "Automatic tracing of decisions to architecture and implementation," in *9th Working IEEE/IFIP Conference on Software Architecture (WICSA'11)*. IEEE, 2011, pp. 46–55.

[27] M. Ali Babar and I. Gorton, "A tool for managing software architecture knowledge," in *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07)*. IEEE, 2007, pp. 11–17.

[28] J. E. Burge and D. C. Brown, "Software engineering using rationale," *Journal of Systems and Software*, vol. 81, no. 3, pp. 395–413, 2008.

[29] N. B. Harrison, P. Avgeriou, and U. Zdun, "Using patterns to capture architectural decisions," *Software*, vol. 24, no. 4, pp. 38–45, 2007.

[30] O. Zimmermann, "Architectural decisions as reusable design assets," *Software*, vol. 28, no. 1, p. 64, 2011.

[31] T.-M. Hesse and B. Paech, "Supporting the collaborative development of requirements and architecture documentation," in *3rd Int. Workshop on the Twin Peaks of Requirements and Architecture*. IEEE, 2013, pp. 22–26.

[32] "Update Site for DecDoc," URL retrieved in 02-2016. [Online]. Available: http://svn.ifi.uni-heidelberg.de/unicase/0.5.2/ures/decdoc-features/

[33] "UNICASE," URL retrieved in 02-2016. [Online]. Available: http://unicase.org/

[34] "EMFStore," URL retrieved in 02-2016. [Online]. Available: http://eclipse.org/emfstore/

[35] T.-M. Hesse, S. Gaertner, T. Roehm, B. Paech, K. Schneider, and B. Bruegge, "Semiautomatic security requirements engineering and evolution using decision documentation, heuristics, and user monitoring," in *First Int. Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE)*. IEEE, 2014, pp. 1–6.

[36] "Papyrus," URL retrieved in 02-2016. [Online]. Available: https://eclipse.org/papyrus/

[37] T.-M. Hesse, A. Kuehlwein, B. Paech, T. Roehm, and B. Bruegge, "Documenting implementation decisions with code annotations," in *27th Int. Conference on Software Engineering and Knowledge Engineering*. KSI Research Inc., 2015, pp. 152–157.

[38] A. van der Hoek, M. Petre, and A. Baker, "Workshop "Studying Professional Software Design" at University of California, Irvine," 2010, URL retrieved in 02-2016. [Online]. Available: http://www.ics.uci.edu/design-workshop/

[39] T.-M. Hesse and B. Paech, "Documenting relations between requirements and design decisions: A case study on design session transcripts," 2016, accepted to appear.

[40] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster, "Reusable architectural decision models for enterprise application development," *Software Architectures*, pp. 15–32, 2007.

[41] M. Razavian, A. Tang, R. Capilla, P. Lago *et al.*, "In two minds: How reflections influence software design thinking," VU University Amsterdam, Tech. Rep., 2015.

[42] R. Capilla, O. Zimmermann, U. Zdun, P. Avgeriou, and J. M. Küster, "An enhanced architectural knowledge metamodel linking architectural design decisions to other artifacts in the software engineering lifecycle," in *Software Architecture: 5th European Conference (ECSA'11)*. Springer, 2011, pp. 303–318.