# Continuous Rationale Visualization

Anja Kleebaum, Barbara Paech
Heidelberg University
Institute for Computer Science
Heidelberg, Germany
{kleebaum, paech}@informatik.uni-heidelberg.de

Jan Ole Johanssen, Bernd Bruegge
Technical University of Munich
Department of Informatics
Garching b. München, Germany
{jan.johanssen, bruegge}@in.tum.de

***Abstract*—Continuous software engineering (CSE) is characterized by frequent changes. It is challenging for developers to change software while sustaining its high quality so that the software is always deployable to the users. Rationale management provides an opportunity to support the change process during CSE. However, rationale management is not well integrated into CSE. Even if the rationale is captured, e. g. in the issue tracking system, it is difficult to access in the context of requirements, code, and other software artifacts. In this paper, we present tool support called ConDec to make rationale explicit during CSE and to integrate rationale into a knowledge graph data structure consisting of requirements, code, and other software artifacts. The knowledge graph is visualized in various ways and the developers can access the knowledge views from software artifacts such as requirements and code. They can interact with and filter the knowledge views. In particular, they can use transitive links, for example, to access all the decisions made in the context of a particular requirement or code file. We demonstrate the usefulness of the knowledge views using a case study project.**

## I. Introduction

Software developers continuously make decisions and need to reflect on former decisions during their work. Their knowledge about decisions is called *decision knowledge* or *rationale* [1], [2]. They make decisions during various software engineering activities, e. g., during the elicitation of requirements, their implementation in code, and testing. Thus, the decisions are not isolated but need to be reflected in the context of the requirements, code, test cases, and other software artifacts. Besides, the decisions depend on each other [3], [4].

*Continuous software engineering* (CSE) needs to be able to handle frequent changes in the decisions and software artifacts [5]. Changes involve the risk to introduce inconsistency through side and ripple effects. Documentation can become outdated or bugs can be introduced in the software. *Rationale management* is a software engineering workflow in which the developers document their decision knowledge and exploit the documentation [1]. In general, rationale management positively influences software development. For example, it promotes knowledge sharing, prevents knowledge vaporization, and supports the change process [6], [7], [8]. Thus, rationale management can be valuable for CSE.

However, rationale management is not well integrated into CSE. That means that CSE lacks systematic techniques and tools for rationale management. In particular, it is not clear to developers how to exploit the decision knowledge documentation [9]. We argue that the developers need good views on the documented decision knowledge in its context.

Our overall goal is to develop techniques and workflows for a *continuous rationale management*. Similar to, for example, unit testing, rationale management should be a well-integrated part of CSE. Lightweight support for explicit rationale management should be integrated into the practices that developers already do, for example, in managing requirements and development tasks in the issue tracking system, in committing code, reviewing and deciding about the acceptance of pull requests, and in conducting meetings.

In [10], we presented ideas on knowledge visualization during CSE. Since then, we develop the ConDec tools [11]. ConDec is open source and comprises extensions for existing software engineering tools[1]. It enables the developers to document decision knowledge in the issue tracking system (ITS) Jira and in the version control system (VCS) git using lightweight annotations [12]. It builds up and visualizes a knowledge graph of decision knowledge, requirements, code, and other software artifacts. We refer to all software artifacts as knowledge elements.

In this paper, we present views provided by ConDec on the knowledge graph and we describe how these views integrate into software development. In summary, the contribution of the ConDec tools is not a single new visualization technique but 1) the visualization of rationale and related knowledge, which is often only implicit and not visualized, and 2) the integration of various knowledge views into CSE.

In the remainder of the paper, we describe overall requirements for continuous rationale management in Section II. In Section III, we describe our solution for these requirements: the knowledge graph including explicit rationale, its views, and how the ConDec users can tailor the views to their needs. In Section IV, we describe a case study project that we used to exemplify the views and to evaluate them. In Section V, we discuss related work and Section VI concludes the paper.

## II. Continuous Rationale Management

We describe the requirements for continuous rationale management, as well as the involved roles and their tasks.

[1]https://github.com/cures-hub; Video about the ConDec views and interaction possibilities: https://se.ifi.uni-heidelberg.de/research/projects/condec.html

Definitive version available at https://doi.org/10.1109/VISSOFT52517.2021.00013

### A. Requirements to support continuous rationale management

In the following, we list high level requirements (R1 – R5) on the ConDec support for rationale management during CSE.

**R1 Decision making support:** *Collaborative, incremental, and rational decision making should be supported.* In practice, decisions are mostly made and documented in a naturalistic way [13], [14]. This means that only a part of the decision knowledge is documented, which makes it hard for other developers to understand the decision and to be convinced of it. Thus, ConDec should enable naturalistic decision making but encourage the developers to collaboratively and incrementally reflect on decisions made. In this way, the decisions and their documentation evolve from being naturalistic to more rational.

**R2 Documentation support:** *Non-intrusive, easy decision knowledge documentation should be supported.* The developers should be able to capture decisions and related decision knowledge in their current development context, e. g., in the issue tracking system, integrated development environment, and during committing. They should not be required to change their development context to capture decision knowledge. Thus, ConDec should enable various documentation locations.

**R3 Exploitation support:** *Non-intrusive, easy usage of decision knowledge should be supported.* The developers should be supported in accessing the documented decision knowledge from various documentation locations in the context of other knowledge (software artifacts). They should be supported in using the knowledge documentation during their daily practices, e. g., when managing requirements and work items in the issue tracking system, when writing code, or when conducting meetings. In particular, the developers should be supported in exploiting the documentation during change for change impact analysis and change execution. ConDec should offer views on the knowledge documentation and support their customization for specific purposes, e. g., change impact analysis. It should support easy knowledge sharing.

**R4 High quality support:** *High documentation quality wrt. consistency and completeness should be supported in an easy way.* The exploitation of knowledge documentation is only useful if the documentation has high quality, i. e., is complete and consistent. The developers should be supported to create and maintain high documentation quality. ConDec should have mechanisms for knowledge quality checking or even enforcement integrated into the development process, e. g., similar to checking and enforcement of unit test coverage.

**R5 Support high amount:** *A high amount of knowledge documentation and exploitation should be supported.* Software development is knowledge-intense and documented knowledge tends to become big with many knowledge elements and links. ConDec needs to be able to tailor the knowledge documentation so that it is useful.

### B. Roles and their tasks

We distinguish two roles that we want to support with the ConDec views: developers and the rationale manager.

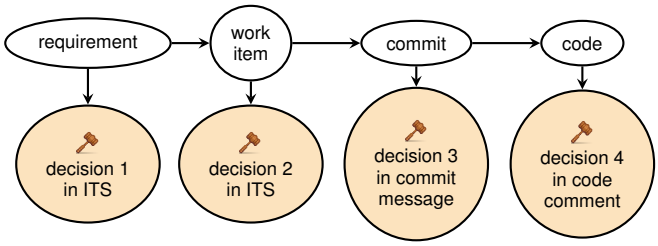



Fig. 1. Schematic illustration of an unfiltered knowledge subgraph. Decisions 1 and 2 are documented in the ITS, e. g., in the description or comments of the requirement and work item, respectively. Decision 3 is captured in a commit message and decision 4 is documented in code comments.

*1) Developers:* Developers could be further refined into roles such as requirements engineers, product owners, software architects, or testers. That means that their tasks are typical tasks for these roles, such as eliciting, documenting, checking, and managing requirements, designing software architecture, implementing, and testing it. However, from our point of view, all of these roles contribute to the development of a software system by making decisions, e. g., on requirements, design, or tests, and they reflect on decisions already made. Thus, they produce and consume decision knowledge. In particular, during the continuous rationale management, they need to document their decision knowledge and exploit the documented knowledge through knowledge visualization.

*2) Rationale manager:* The rationale manager is a role whose task is to set up the continuous rationale management process, e. g., by teaching the developers how to document and exploit decision knowledge [15]. Another task of the rationale manager is to monitor the rationale documentation and to check and assure its high quality. For example, the rationale manager checks that the documented decision knowledge is complete, consistent, and linked to other knowledge.

## III. Knowledge Graph with Explicit Rationale

In this section, we describe the knowledge graph data structure, how it is presented, and functionality to customize the views to specific purposes, e. g. through transitive linking.

### A. Knowledge graph data structure

The ConDec views work on a directed graph data structure that we refer to as *knowledge graph*. The knowledge graph consists of knowledge elements, i. e., software artifacts, and relationships. Both, the knowledge elements and the relationships can be of various types. High-level types of knowledge elements are decision knowledge, system knowledge, and project knowledge [16]. Decision knowledge elements cover decision problems, i. e., issues to solve, alternative solution options, decisions, pro-, and con-arguments. We use the decision documentation model for flexible documentation of decision knowledge [17]. System knowledge elements cover requirements, code, and test cases, whereas project knowledge elements cover development tasks (work items), commits, pull requests, and other process-related artifacts.

Developers document decision knowledge in different documentation locations, in particular, in the description and comments of tickets in the ITS, commit messages, code comments, or as entire tickets with specific types [12]. This enables the developers to easily document decision knowledge within their current development context (R2). Regardless of their origin and documentation location, ConDec automatically adds the decision knowledge elements to the knowledge graph.

Relationships, i. e., links or graph edges, can be established between the knowledge elements of all types in the knowledge graph. The developers or the rationale manager can manually link knowledge elements, but ConDec also offers the following mechanisms for automatic link creation and maintenance: 1) ConDec automatically links tickets such as development tasks, bug reports, or requirements in Jira with code in git if the code was changed in a commit that contains the ticket identifier in its commit message. Providing ticket identifiers in commit messages established as a common convention [18]. 2) ConDec automatically links decision knowledge elements documented in Jira issue text (description or comments), commit messages, and code comments to other elements. For example, ConDec automatically links a decision problem to the respective ticket (development task, bug report, or requirement). It links the solution options to the decision problem and arguments to solution options according to their sequential order in the text. Relationship types between solution options, i. e., between decisions and alternatives, can be the following: enables, constrains, forbids, comprises, subsumes, overrides, and relates [3]. In addition, pro-arguments *support* solution options, whereas con-arguments *attack* solution options. The rationale manager can add custom element and link types.

Figure 1 sketches knowledge elements of different types and their links in the knowledge graph. In this schematic illustration, the decisions are the only decision knowledge elements shown. Other decision knowledge elements, such as the decision problems, alternatives, and arguments, should be documented to detail the decisions.

We use the jGraphT library to implement the knowledge graph data structure. Next to the data structure, this library offers useful graph algorithms, such as for finding shortest paths, which we use for transitive linking (Subsection III-C).

### B. Views on the knowledge graph data structure

In the following, we list the views on the knowledge graph (V1 – V7) with a short description and a usage example. All the views use the same knowledge graph data structure underneath, which enables to easily add new views. Note that the views mostly show only parts of the entire knowledge graph data structure for a project. We refer to such a part as *knowledge subgraph*. The knowledge graph views are the building blocks for higher-order support and views integrated into software development. Table I gives an overview of the roles, their tasks, and the respective ConDec support.

**V1 Node-link diagram:** *ConDec visualizes the knowledge (sub-)graph as a node-link diagram.* Figure 2 shows an example of the node-link diagram including an epic (*Search Processing*, colored in blue), six user stories (e. g., *As a search engine user, I want to find documents that contain synonyms to my search terms, so I don't have to try each synonym individually*), code, and decision knowledge elements (also colored). When documenting requirements using a hierarchical requirements model, e. g., using epics and user stories, it is interesting to see the user stories linked to the epic to understand the epic. ConDec explicitly presents the decision knowledge elements in addition to the requirements. Code files are linked through the tool to work items (development tasks) and the work items are linked to user stories. This is reflected by links in the knowledge graph data structure. In Figure 2, code files are transitively linked to user stories (omitting the work items for better understanding). We use the vis.js network library to create the node-link diagram.

TABLE I
ROLES, THEIR TASKS, OVERALL REQUIREMENTS, AND CONDEC SUPPORT

| Role | Task and Requirement | ConDec support | Views |
|---|---|---|---|
| Developer | Collaboratively make and share decisions, R1, R3 | Rationale backlog listing unresolved decision problems | V1, V2, V4 |
| | | Meeting agenda with explicit decision knowledge | V3 |
| | | Release notes with explicit decision knowledge | V3 |
| | | Chronology view plotting recently documented and updated knowledge elements | V5 |
| | Document decision knowledge and exploit knowledge documentation, R2, R3, R5 | Jira issue view to see and interact with knowledge graph views from specific requirement or other Jira issue | V1, V2, V4, V5, V7 |
| | | IDE extensions to navigate from code file to knowledge graph views centered around this code file in Jira, F5 | V1, V2, V4, V7 |
| | | CIA support through change impact highlighting, F6 | V1, V2, V3, $V4_{adj}$ |
| | | Exploitation support through transitive linking, F2 | V1, V2, V3, $V4_{adj}$, V6 |
| | Continuously improve knowledge quality, R4 | Quality/DoD checking result visualization using nudging mechanisms, F7 | V1, V2, V3, V4, V5, V7 |
| Rationale manager | Assure high documentation quality, R4 | Dashboard for rationale coverage, intra-rationale completeness, checking of DoD, and for other metrics | V6 |
| | | Rationale backlog listing knowledge elements that violate the DoD | V1, V2, V4 |

**V2 Knowledge tree view:** *ConDec visualizes knowledge subgraphs starting from a selected knowledge element as a tree. The selected element is the root in the tree.* ConDec provides two different visualizations: Figure 3 shows an *indented outline* visualization ($V2_{ind}$) starting from a user story. Figure 4 shows a *node-link tree diagram* ($V2_{nld}$) starting from a decision problem (issue). ConDec's knowledge tree views are created using the jsTree and treant libraries, respectively.

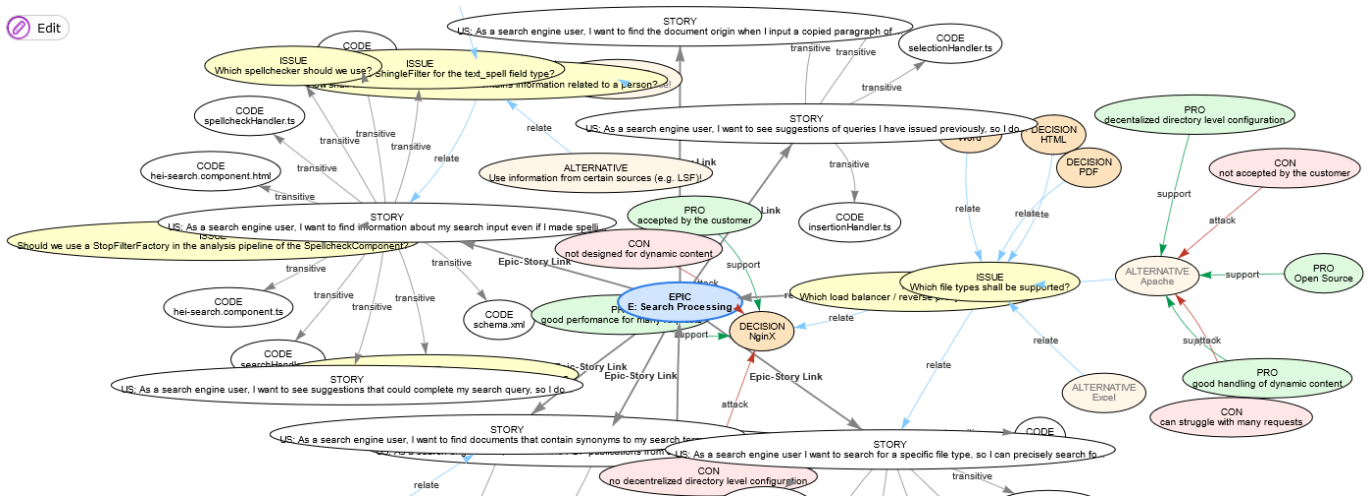



Fig. 2. Node-link diagram (V1) showing the context of an epic. The subgraph shows user stories, code, and decision knowledge reachable from the epic in a link distance of 3 in the knowledge graph. Transitive links between user stories and code replace filtered-out work items (development tasks).

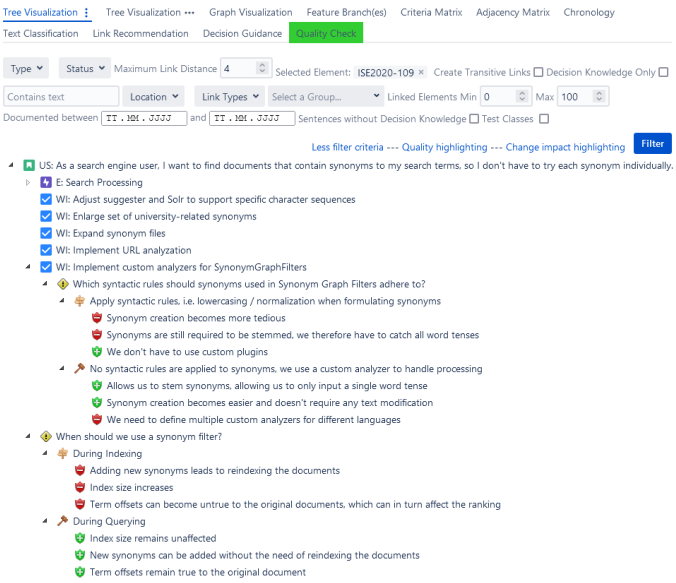



Fig. 3. Knowledge tree view ($V2_{ind}$) with a user story being the root element. The user story is linked to an epic (E), work items (WI), and decision knowledge. Decision knowledge is also transitively connected to the user story via the link to the work item.

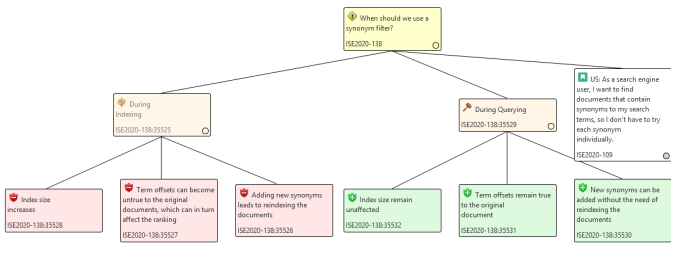
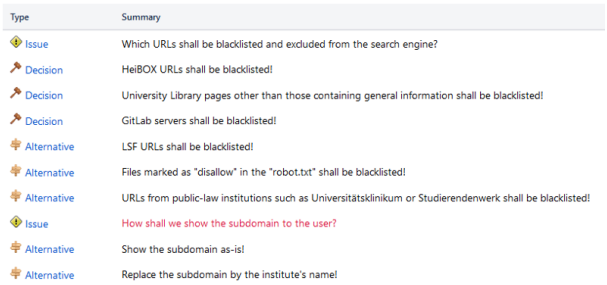



Fig. 4. Knowledge tree view ($V2_{nld}$) starting from a decision problem (issue). The decision problem is linked to a user story. The user story is also linked to further knowledge elements, but the subtree is collapsed.



| Type | Summary |
|---|---|
| Issue | Which URLs shall be blacklisted and excluded from the search engine? |
| Decision | HeiBOX URLs shall be blacklisted! |
| Decision | University Library pages other than those containing general information shall be blacklisted! |
| Decision | GitLab servers shall be blacklisted! |
| Alternative | LSF URLs shall be blacklisted! |
| Alternative | Files marked as "disallow" in the "robot.txt" shall be blacklisted! |
| Alternative | URLs from public-law institutions such as Universitätsklinikum or Studierendenwerk shall be blacklisted! |
| Issue | How shall we show the subdomain to the user? |
| Alternative | Show the subdomain as-is! |
| Alternative | Replace the subdomain by the institute's name! |

Fig. 5. List view (V3) used as a stand-up table as part of a meeting agenda. The unresolved decision problem (issue) is highlighted using red text color to nudge the developers to collaboratively make and document a decision.

The knowledge graph can contain cycles. These cycles need to be removed when converting the graph into a tree. Figure 8 shows that cycles are handled by duplicating knowledge elements in the tree views. In this example, a cycle is established because the solution option *Standard Parser* for the issue *Which Solr query should we use?* has two con-arguments that are both linked to the same quality requirement (QR).

**V3 List view:** *ConDec visualizes (parts of) the knowledge graph as a list of knowledge elements.* The list view can be integrated as a stand-up table into meeting agendas in the wiki using the ConDec Confluence extension so that the developers can discuss recently made decisions and open decision problems during meetings (Figure 5). Besides, ConDec supports the creation of release notes including explicit decision knowledge for knowledge sharing.

**V4 Adjacency and criteria matrix view:** *ConDec visualizes (parts of) the knowledge graph as an adjacency matrix. For decision problems, ConDec shows their solution options (alternatives and the decision), arguments, and criteria as a criteria matrix.* Figure 6 shows a similar knowledge subgraph as shown in Figure 4 as an adjacency matrix (only decision knowledge is shown). A cell of the matrix is colored if

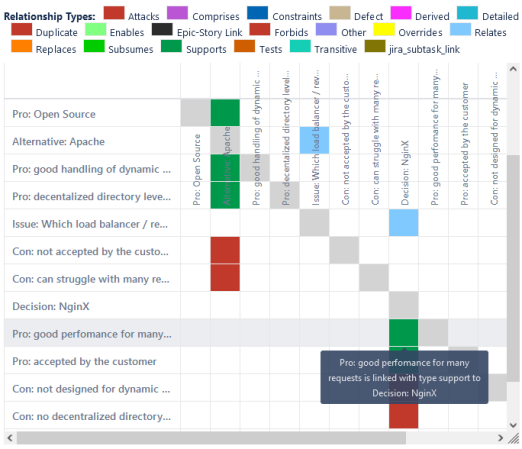



Fig. 6. Adjacency matrix view ($V4_{adj}$) for a decision problem. The matrix cells are colored if there is a directed link (direction is important). The color indicates the link/relationship/edge type in the knowledge graph.

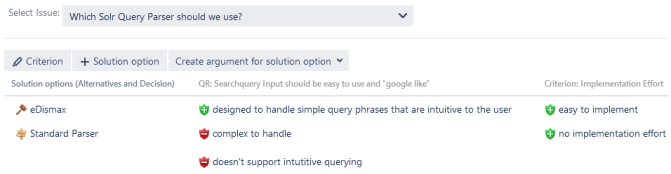



Fig. 7. Criteria matrix view ($V4_{cri}$) for a decision problem including solution options (alternative and the decision), arguments, and criteria.

there is a directed link from a knowledge element in the row to another knowledge element in the column. Link type colors are fixed for default link types such as *attacks* (red) and *supports* (green). To enable a flexible knowledge meta-model, ConDec uses the hash value of custom link types such as *Epic-Story Link* as the color. However, these hash colors might not be intuitive. Figure 7 shows a criteria matrix for a different decision problem. This view visualizes criteria used during decision making as columns. Criteria can be QRs or other aspects such as the implementation effort. The node-link diagram (V1) or tree view (V2) can then be used to see all decisions that support or attack a certain QR or other criterion (Figure 8). For ConDec's matrix views, we decided not to use an external library but created it from scratch. This was the easiest way to add the features described in Subsection III-C.

**V5 Chronology view:** *ConDec visualizes the knowledge elements of the knowledge graph in chronological order depending on their creation time or time of last change/update.* Figure 9 shows decisions documented at the beginning of our case study project (see Section IV) in chronological order. In addition to showing decisions only, the chronology view can be used to plot other knowledge elements at their date of creation or the last update, e. g., requirements, code files, or other decision knowledge elements. The knowledge elements are either placed on a specific date (as shown in Figure 9) or they range from their creation date to the date of the last update depending on the filter criteria (Subsection III-C). ConDec's chronology view is created with the vis.js timeline library.

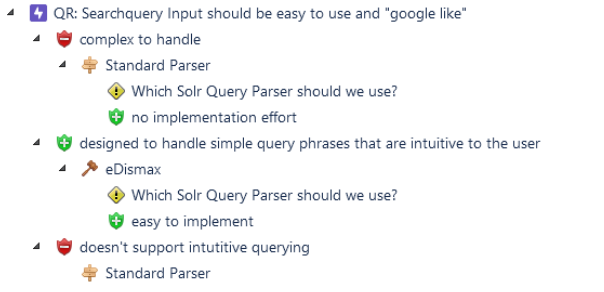



Fig. 8. Knowledge tree view ($V2_{ind}$) for decision knowledge linked to a quality requirement, which was one criterion for decision making in Figure 7.

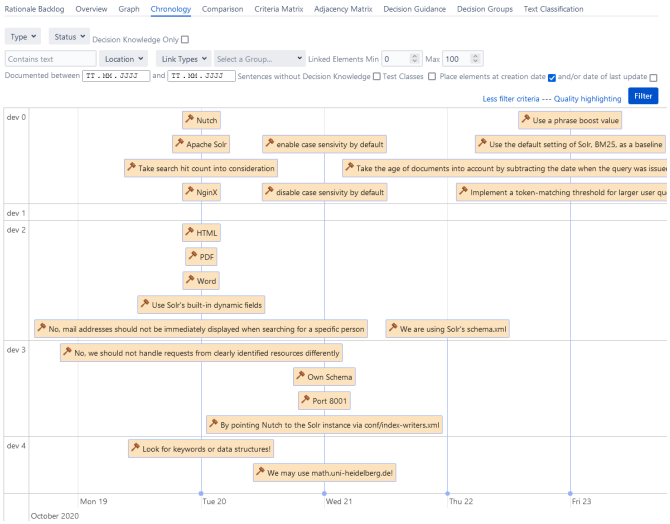



Fig. 9. Chronology view (V5) of decisions made at the beginning of the case study project. The x-axis shows the documentation date of the decisions. On the y-axis, the decisions are grouped according to the developers (anonymized) who documented them. Filtering criteria are shown at the top.

**V6 Metrics view:** *ConDec visualizes metrics calculated on the knowledge graph data structure in a dashboard, e. g. using pie charts and box plots.* Figure 10 shows a box plot and a pie chart as an excerpt of the ConDec dashboard. ConDec's metric views are created using the Apache ECharts library.

**V7 Detail view:** *This view shows the details of a specific knowledge element, i. e., its attributes and meta-data.* For example, details can be the author, the description, the time

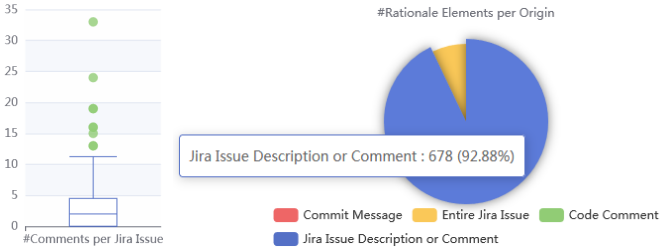



Fig. 10. Metrics view (V6). **Left**: Box plot to present the number of comments per Jira issue. **Right**: Pie chart to give an overview where rationale elements are documented.

TABLE II
FEATURES AVAILABLE IN KNOWLEDGE GRAPH VIEWS

| View/Feature | Filtering, F1 | Transitive linking, F2 | Change execution, F3 | Spec. level of detail, F4 | Navigation, F5 | CIA, F6 | DoD, F7 |
|---|---|---|---|---|---|---|---|
| Node-link diagram, V1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Tree, V2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| List, V3 | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |
| Adjacency matrix, $V4_{adj}$ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |
| Criteria matrix, $V4_{cri}$ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |
| Chronology, V5 | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ |
| Metrics, V6 | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ |
| Detail, V7 | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |

of creation and the last update, and other attributes. The developers can use the views of the tools that ConDec extends, e. g., the Jira issue view (Figure 15) or the code file view of the integrated development environment (IDE) to see details of a specific knowledge element.

### C. Features: Filtering, interaction, and highlighting

The views on the knowledge graph data structure can be adapted to specific purposes through filtering, highlighting, and interaction. In the following, we list these features (F1 – F7) with a short description and a usage example. Similar to the goal that all views share the same data structure, our goal was to support the same features for all the views to enable easy rationale and knowledge management. However, there are some differences due to the nature of the view (Table II).

**F1 Knowledge graph filtering:** *Developers filter the knowledge graph using various filter criteria.* For instance, filter criteria are the knowledge type, status (e. g., resolved, unresolved, decided, rejected), documentation location (Jira issue, Jira issue text, commit message, code comment), number of hops/link distance, node degree (min and max number of links), textual content, time, and decision levels [19] or groups [20]. These basic filtering possibilities are the same for all views that show more than one element, but some views provide extra filter criteria, e. g., the chronology view (V5). Figure 3, Figure 9, and Figure 12 show filtering possibilities.

**F2 Transitive linking:** *Developers exploit transitive links between knowledge elements.* For example, they examine all decisions made in the context of an epic. The decisions can be documented in user stories, work items, commit messages, and code files reachable from the epic. Examples for transitive linking are shown in Figure 2 and Figure 12. Note that the transitive links in the knowledge graph do not necessarily create a transitive closure, i. e., only the elements that would not be reachable are transitively linked (Figure 11). Algorithm 1 sketches the algorithm that we use to create transitive links.[2]

[2]https://github.com/cures-hub/cures-condec-jira/blob/master/src/main/java/de/uhd/ifi/se/decision/management/jira/filtering/FilteringManager.java

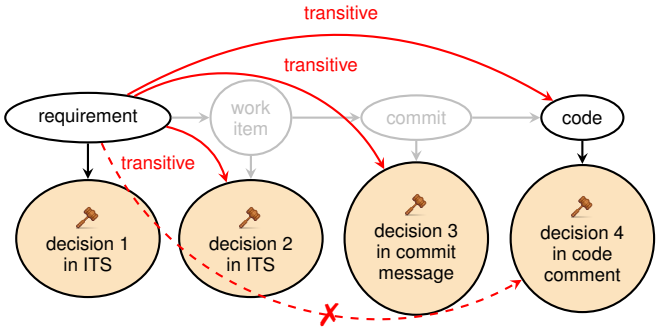



Fig. 11. Schematic illustration of transitive linking. In this example, the gray-colored knowledge elements and links are filtered out. The red solid lines represent transitive links that replace filtered-out elements. The dashed line indicates a link that would be necessary to create a transitive closure. This link is not created in ConDec because all the elements are already reachable.

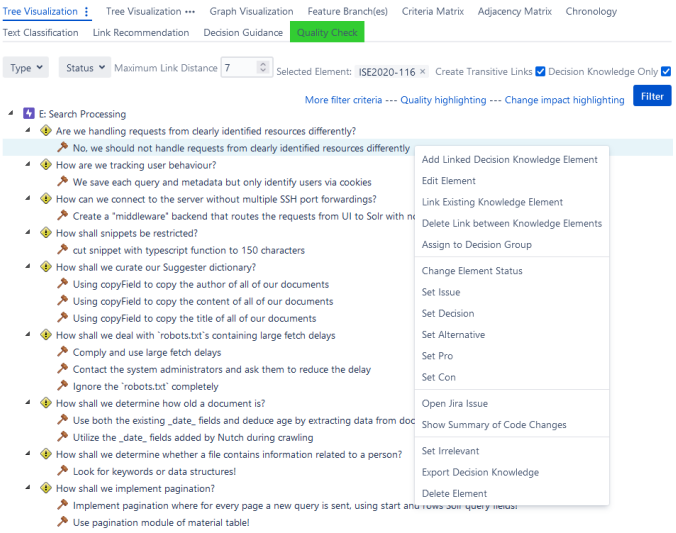



Fig. 12. Epic with decision problems (issues) and decisions that are transitively linked to the epic within the link distance 7 in the knowledge graph data structure. Filtering possibilities and the context menu are shown. The actual tree is longer and also contains the decisions in Figure 3.

**F3 Change execution:** *Developers create, update, and delete knowledge elements and links within the views of the knowledge graph data structure.* For example, they can add new decisions for the implementation of a requirement using a *context menu* as shown in Figure 12. They can add and delete links, as well as update link types using *drag and drop* or by clicking a matrix cell (V4). Some visualization libraries such as vis.js offer additional manipulation possibilities.

**F4 Specifying the level of detail:** *Developers change the level of detail to either understand the big picture (e. g. how knowledge elements relate to each other) or to see details (e. g. a summary of a particular knowledge element).* The node-link diagram (V1) and chronology view (V5) allow to specify the level of detail by *zooming* in and out. Besides, parts of the view can be *collapsed* in the node-link diagram (V1), tree (V2, Figure 4), and metrics (V6) views.

**Algorithm 1:** Transitive linking, i. e. replacement of filtered-out knowledge elements on a graph path with links

```
Input: graph, selectedElement, linkDistance, filterCriteria
Result: filtered graph in that filtered-out knowledge elements (nodes/vertices) are replaced with transitive links (edges/relationships)
singleSourcePaths ← findAllShortestPaths(graph, selectedElement, linkDistance)   // find all shortest paths starting from the selected element in the unfiltered knowledge graph within the link distance e.g. using Dijkstra
filteredGraph ← createFilteredGraphThatMatchesFilterCriteria(graph, filterCriteria)   // filter knowledge graph according to filter criteria, e.g., element type, status, documentation location, node degree, decision levels
for element : filteredGraph.vertexSet() do   // iterate over the remaining elements in the filtered graph
    path ← singleSourcePaths.getPathTo(element)   // get path in the unfiltered knowledge graph
    lastUnfilteredElementOnPath ← selectedElement   // remember visited element on path that is not filtered out
    for elementOnPath : path.getVertexList() do   // iterate over elements on the path in the unfiltered graph
        if !filteredGraph.vertexSet().contains(elementOnPath) then   // the element on the former path is filtered out
            continue   // keep on walking along the path until we find an element not filtered out
        else   // the element on the former path is still existing in filtered graph
            if !filteredGraph.containsEdge(lastUnfilteredElementOnPath, elementOnPath) then   // check whether link already exists
                transitiveLink ← new Link(lastUnfilteredElementOnPath, elementOnPath, LinkType.TRANSITIVE)
                filteredGraph.addEdge(transitiveLink)   // add new transitive link between elements on the same path
            lastUnfilteredElementOnPath = elementOnPath   // remember visited element on path that is not filtered out
```

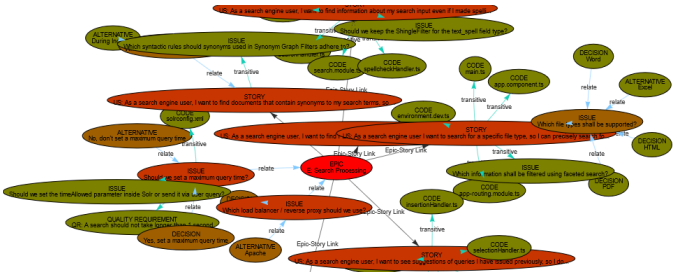


Fig. 13. Node-link diagram (V1) with change impact highlighting. Decision problems (issues), solution options (decisions and alternatives), requirements, and code files are shown that might be impacted by a change in the epic. The color indicates the likelihood of change impacts: red elements are probably more impacted by a change than green elements.

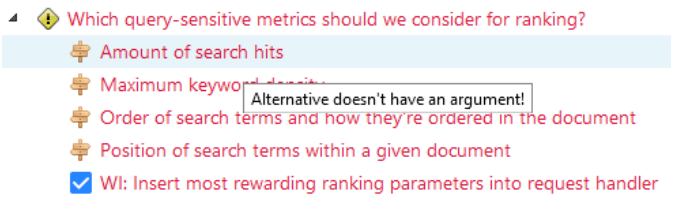



Fig. 14. Knowledge tree view ($V2_{ind}$) with knowledge elements violating the DoD and quality highlighting. The decision problem (issue) is colored in red because it is unresolved, i. e., a decision needs to be made. The alternatives are colored in red because no arguments are linked, i. e., a criterion for intra-rationale completeness is violated (as indicated by the tooltip). The work item (WI) is colored in red because its decision coverage is too low.

**F5 Integrated navigation:** *Developers navigate to other parts of the knowledge graph and to different knowledge graph views.* For example, the navigation is enabled through the context menu (Figure 12), through hyperlinks on UI elements, such as menu items and matrix headers (V4), or by clicking on data points in the metrics plots (V6). ConDec also provides extensions for IDEs (Eclipse and Visual Studio Code) that enable to navigate from code files to the knowledge graph views in Jira with the code file being the selected element.

**F6 Change impact highlighting:** *When changing a knowledge element, developers estimate which other knowledge elements need to be changed as well.* The node-link diagram, tree, list, and matrix views (V1 – V4) can be used for change impact analysis (CIA). ConDec colors the knowledge elements in these views according to the likelihood that they are affected by a change in the selected element (Figure 13). ConDec combines different CIA algorithms, in particular, a traceability and a rule-based approach [21], [4].

**F7 Quality highlighting:** *Developers see which knowledge elements violate or fulfill the definition of done (DoD). If the DoD is not fulfilled, they see which DoD criteria are violated.* ConDec introduces a DoD for knowledge documentation. Criteria of the DoD are the rationale coverage (see Figure 16), the intra-rationale completeness (see Figure 17) and other criteria (e. g., that small code files and code files for unit testing are not checked). ConDec checks whether the DoD is fulfilled for knowledge elements in the knowledge graph and visualizes the results of the check. There are various ways to visualize the results of DoD checking. Figure 3 and Figure 12 show the *Quality Check* tab which is colored in green if the DoD is fulfilled. If the DoD is violated, this tab is colored in orange (if some criteria are fulfilled) or in red (if no criterion is fulfilled). Besides, the knowledge elements that violate the DoD are highlighted with a red text color within the knowledge graph views (Figure 14) to indicate quality problems. Tooltips explain which DoD criteria are violated. The indication of quality problems through text coloring is done in V1 – V5. This should nudge the developers to improve the DoD using ambient feedback nudging mechanisms [22].

ConDec also offers a rationale backlog with preset filter criteria so that only knowledge elements that violate the DoD are shown. In particular, the rationale backlog shows open decision problems for which a decision still needs to be made (or documented), challenged decisions, and all knowledge elements that violate the DoD. We did not list the rationale backlog as a separate knowledge graph view because it comprises the knowledge graph views but with special filtering.

## IV. Case Study and Evaluation

We applied ConDec during a student project that simulated a CSE project in industry. The screenshots in this paper were taken in this project to demonstrate the view usage in a realistic context. In this section, we provide a brief description of this project and a first evaluation of the view usage.

### A. Project description

The goal of the project was to develop a web search engine for the websites of Heidelberg University. The project started in October 2020 and finished in March 2021. There were two block courses at the beginning and end of the project lasting fourteen days, respectively. During these block courses, the students worked full-time on the project. We introduced the students to rationale management and the ConDec tools using a lecture on rationale management [15] at the beginning of the project. During the semester, the students worked part-time on the project. Five students were involved in the project. The project followed the Scrum process with sprints lasting from two weeks during the block courses to four weeks during the semester. During every sprint, one student took the role of the Scrum Master and was responsible for the sprint review that was held at the end of every sprint. At the same time, this student also took the role of the rationale manager. All other students took the role of developers.

Next to the tasks listed in Subsection II-B and Table I, we had the following major process requirements regarding rationale management: 1) The rationale manager's task was to present explicit decision knowledge, i. e., decisions made and open decision problems during the sprint review. For that purpose, the developers should document decision knowledge during the sprint using ConDec. 2) The rationale backlog should only include unresolved decision problems at the end of the sprint but no other knowledge elements that violate the DoD so that the documentation quality is high. 3) The students should include a stand-up table with decision knowledge into agendas and protocols of their meetings (Figure 5). They should create release notes with decision knowledge at the end of every sprint.

The students used the issue tracking system Jira to document requirements, development tasks, and bug reports. They committed code to work items or to bug reports mentioning the respective ticket identifier in the commit messages. They had regular meetings with the customer who was a University professor from another group. They elicited the requirements with the customer and documented them, e. g., in the form of three epics titled *Search Input*, *Search Processing*, and *Search Result Presentation* and 13 user stories. The requirements elicitation, documentation, checking, and management was an ongoing, iterative process, i. e., the requirements documentation was not fixed from the beginning, but for simplicity, we only report the results. Next to the requirements, the developers created tickets for major decision problems (issues) that they faced, for example, *Which framework should we use for the search engine?* and *Which framework should we use as a web-crawler?* They captured solution options and arguments in the description and comments of these issues. Later, they linked the decision problems to requirements or work items, e. g., to *Install Solr* and *Install Nutch* to make the knowledge elements reachable within the knowledge graph. Besides documenting decision problems as tickets (i. e. as entire Jira issues), the developers also captured decision problems and other decision knowledge elements in the text of requirements, work items (Figure 15), and bug reports. They explored the documented decision knowledge in the context of other knowledge using the knowledge graph views (Section III).


Fig. 15. Decision knowledge captured in the description of a work item.

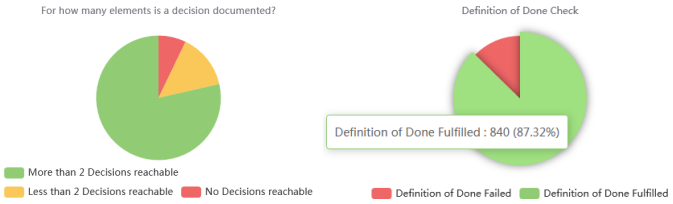



Fig. 16. **Left**: Pie chart to present the coverage of requirements and code files with documented decisions within a link distance (number of hops) of 3 in the knowledge graph data structure of the case study project. **Right**: Pie chart to present the DoD fulfillment and violation for all knowledge elements in the project. The decision coverage is one criterion for the DoD.

In total, the students documented the following system and project knowledge elements: Three epics, 13 user stories, six quality requirements, one user role, four personas, 132 work items (development tasks), 34 bug reports, and 658 code files of types ts, html, and xml. They documented 729 decision knowledge elements in two different documentation locations: 51 entire Jira issues (mainly for decision problems/issues) and 678 in the description or comments of existing Jira issues such as user stories or development tasks (Figure 10). Figure 17 shows the distribution of the decision knowledge types and one aspect of the intra-rationale completeness measured after the project was finished. In total, 68 decision problems (issues), 102 decisions, 116 alternatives, and 443 arguments (either pros or cons) were documented.

### B. Measurement of view usage

This subsection describes an initial study to evaluate the ConDec views. Our goal was to get an impression of which views the developers and the rationale manager prefer.

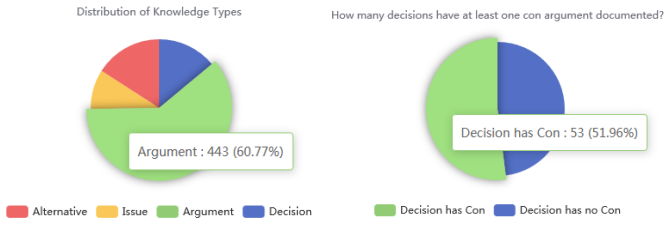



Fig. 17. **Left**: Distribution of rationale types. **Right**: Visualization of one criterion (con-arguments against decisions) of the intra-rationale completeness.

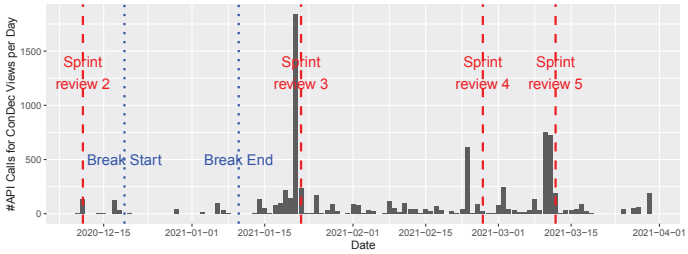



Fig. 18. View usage approximated using the total number of REST API calls per day during the case study project.

*1) Data collection:* The ConDec implementation uses a REST API for the creation of the knowledge graph views. We logged the usage of this REST API. The students knew that we monitor the ConDec usage for evaluation purposes but we told them that this does not influence the grading. We started the data collection after the second sprint because then the students seemed to be familiar with rationale management and the ConDec views. At the end of the project, we interviewed them to ask them for their attitude towards the ConDec views.

*2) Data analysis:* We analyzed the usage data of the REST API using an R-Script[3]. We anonymized the collected data but ensured that we only analyze the REST API usage of the project participants. We analyzed the total usage over time and also the total usage per view.

*3) Results and discussion:* Figure 18, Figure 19, and Table III show the results of the view usage measurement.

Figure 18 shows that the total amount of view usage varied during the project. In particular, before the sprint reviews, the view usage seemed to be higher. A reason for this might be that the participants had to tidy up their knowledge documentation to present it to the customer during the sprint review. For that purpose, they used the ConDec views. During the Christmas break, there was only little view usage.

Figure 19 shows that the view usage differed between the ConDec views and also between the developers, which are called dev 0 – dev 4. Table III shows the total amount of clicks per view. The knowledge tree views (V2) were most often used (6890 number of REST API calls). More precisely, the indented outline $V2_{ind}$ (Figure 3) was approximately used twice as often as the node-link tree diagram $V2_{nld}$ (Figure 4). When asking the project participants for their attitude towards the ConDec views, they confirmed that the indented outline

[3]https://github.com/cures-hub/cures-condec-jira/tree/master/doc/logging/

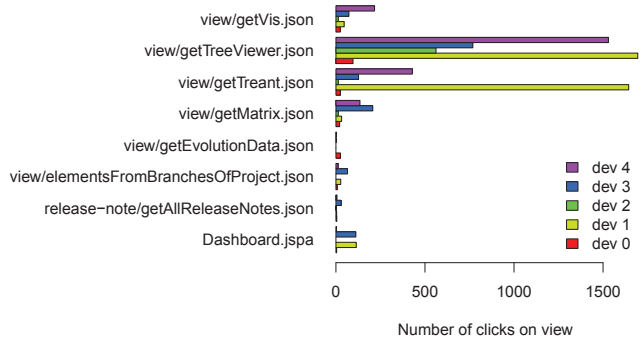



Fig. 19. View usage approximated using the total number of REST API calls per developer (anonymized) during the case study project.

was most useful because it provided the best overview. Regarding the node-link tree diagram, they criticized that they often had to scroll a lot, which they found not useable. The project participants used the node-link diagram (V1) and adjacency matrix ($V4_{adj}$) similarly often. They stated that both views are useful to see and manage links, i. e., their types and directions. Link types and directions are not shown in knowledge tree views. They emphasized that they liked the colorful presentation of link types in the adjacency matrix view. However, they stated that both the node-link diagram and adjacency matrix were not suitable to get a good overview of the documented knowledge. The metrics plots (V6) were accessed through the dashboard Servlet API, which plots various metrics at once (for decision coverage, intra-rationale completeness, DoD checking, and other general metrics such as distribution of types and documentation locations of decision knowledge). The project participants stated that they did not use the dashboard and metrics plots very often because they used the rationale backlog which lists all knowledge elements that violate the DoD instead. The chronology view (V5) was not used very often (only 29 clicks in total). When asking the project participants about their attitude, they said that they did not use the chronology view because they already included a list of decision knowledge elements (V3) in their meeting agenda and used it as a stand-up table. They found that the chronology view could replace the list in meeting agendas, but that using both the list and the chronology view is not necessary. Similarly, they stated that they only created the release notes listing all knowledge elements documented for a

TABLE III
TOTAL NUMBER OF CLICKS PER CONDEC VIEW

| Knowledge Graph View | #Clicks | Name of REST/Servlet API |
|---|---|---|
| Node-link diagram, V1 | 373 | view/getVis.json |
| Tree, $V2_{ind}$ | 4652 | view/getTreeViewer.json |
| Tree, $V2_{nld}$ | 2238 | view/getTreant.json |
| Adjacency matrix, $V4_{adj}$ | 408 | view/getMatrix.json |
| Criteria matrix, $V4_{cri}$ | 0 | view/getDecisionTable.json |
| Chronology, V5 | 29 | view/getEvolutionData.json |
| Metrics, V6 | 233 | Dashboard.jspa |

TABLE IV
TOOLS FOR RATIONALE MANAGEMENT AND THEIR VIEWS

| Tool | Node-link, V1 | Tree, V2 | List, V3 | Matrix, V4 | Chronology, V5 | Metrics, V6 | Detail, V7 | Other |
|---|---|---|---|---|---|---|---|---|
| ConDec | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | (✓) | ✗ |
| SEURAT [23], [24] | ✗ | ✓ | ✗ | ✓ | ✗ | (✓) | ✓ | quality profile report, rationale task list |
| ADDSS [25], [26] | ✓ | ? | ✓ | ? | ✓ | ✗ | ✓ | stakeholder involvement |
| Decision Architect [27] | ✓ | ✓ | ? | ✓ | ✓ | ✗ | ✓ | stakeholder involvement |
| Kruchten's ADD Ontology Tool [28] | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | relationship list |
| Ontology-Driven Visualization Tool [29] | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ? | effect matrix |
| Compendium-based ADD Tool [30], [31] | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | |

sprint because they were required to create it but did not use it afterward. However, the release notes could be interesting for the customer, but not all decisions should be included then. The project participants did not use the criteria matrix ($V4_{cri}$) at all. They stated that, in general, they find it useful to consider criteria such as quality requirements during decision making, but they did not find it necessary in their project. However, they linked work items to quality requirements and used other knowledge graph views to inspect the knowledge documentation in the context of the quality requirements.

*4) Threats to validity:* The number of API calls is one indicator for view usage but needs to be assessed with caution: As shown in Table I, the knowledge graph views V1 – V7 are the building blocks of support for a continuous rationale visualization. In particular, the knowledge tree view $V2_{ind}$ lists the knowledge elements with violating DoD in the rationale backlog. Thus, whenever the developers accessed the rationale backlog, the REST API of $V2_{ind}$ was called. There is no default view when accessing the knowledge graph from Jira issues such as requirements or work items. However, the ConDec Jira plug-in remembers the view that a developer selected and makes it the default until the browser session is finished.

## V. RELATED WORK

In this section, we present related work regarding 1) the visualization of rationale in its context to other knowledge and 2) the integration of rationale management into CSE.

Table IV lists tools that visualize rationale and compares their views with the ConDec views (V1 – V7). While most of the existing tools focus on architectural design decisions (ADDs), ConDec does not restrict the kind of decisions to be captured and visualized. Shahin *et al*. provide an overview of tools that support visualization of ADDs [30]. Besides, they assess whether the Compendium tool can be used to visualize ADDs. SEURAT inspired our dashboard (V6) development and the DoD criteria since it infers over the rationale to look for potential problems. It offers a rationale task list.

The visualization of rationale in its context to other knowledge is closely related to traceability visualization. For example, Filho and Zisman present the D3TraceView tool, which offers similar views as ConDec (matrix, list, and tree), but also a radial tree and sunburst view [32]. Bacher *et al*. develop different tree visualizations for source code comprehension, in particular, a circular tree-map and an icicle tree [33]. Kugele and Antkowiak use the metaphor of an *impact city* to visualize CIA results [34]. These works do not focus on rationale.

Yang *et al*. [35] discuss agile practices that can be integrated into knowledge management: backlog, iterative and incremental development, refactoring, continuous integration, effective communication, and just enough work. In particular, the idea of managing a *decision backlog* has been suggested by various authors, e. g. [36], [37], [38], [39]. In addition, ConDec also shows the requirements and code files that violate the DoD regarding knowledge documentation in the rationale backlog.

New aspects about ConDec are that it integrates into the practices and tools that the developers use (e. g., into managing requirements in the ITS). ConDec supports various documentation locations for rationale. None of the existing tools seem to support transitive link visualization (F2).

## VI. CONCLUSIONS AND FUTURE WORK

Rationale management is beneficial for the change process during fast and rapid software development processes such as CSE. However, decision knowledge is often implicit and not visualized during CSE. Based on a knowledge graph data structure, we presented tool support called ConDec to make decision knowledge explicit. We elicited five high level requirements R1 – R5 that address rationale management during CSE. We designed seven views V1 – V7 on the knowledge graph and seven features F1 – F7 to tailor the views for specific purposes. In a case study that simulated an industry project in an academic setting, we demonstrated the usefulness of ConDec's views and its features. In the future, we will do a further evaluation on the usefulness of ConDec. In particular, the visualization of transitive links can be helpful to exploit the distributed knowledge documentation during CSE. However, this needs further evaluation. Besides, ConDec's knowledge graph data structure, its views, and its features should be further improved. Mechanisms for trace link improvement and maintenance should be added [18]. Change impact estimation should combine more approaches [21], [4], its visualization could be done e. g. using *impact cities* [34], and automatic change execution could be supported. We will work on further nudging mechanisms [22] to trigger rationale management.

## ACKNOWLEDGEMENT

This work was partly supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution (CURES

project). We thank the students who supported the development of the ConDec tools and the project participants as well as Doris Keidel-Mueller and the anonymous reviewers.

## REFERENCES


[1] A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, *Rationale Management in Software Engineering: Concepts and Techniques*. Springer, 2006.

[2] J. E. Burge and D. C. Brown, "Software Engineering Using RATionale," *Journal of Systems and Software*, vol. 81, no. 3, pp. 395–413, 2008.

[3] P. Kruchten, "Documentation of software architecture from a knowledge management perspective – design representation," in *Software Architecture Knowledge Management*. Springer, 2009, pp. 39–57.

[4] C. Carrillo and R. Capilla, "Ripple effect to evaluate the impact of changes in architectural design decisions," in *12th European Conf. on Software Architecture (ECSA'18)*. Madrid, Spain: ACM, 2018, pp. 1–8.

[5] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017.

[6] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, "10 years of software architecture knowledge management: Practice and future," *Journal of Systems and Software*, vol. 116, pp. 191–205, 2016.

[7] L. Bratthall, E. Johansson, and B. Regnell, "Is a Design Rationale Vital when Predicting Change Impact? – A Controlled Experiment on Software Architecture Evolution," in *2nd International Conference on Product Focused Software Process Improvement (PROFES)*. Oulu, Finland: Springer, 2000, vol. 1840 of LNCS, pp. 126–139.

[8] J. Cleland-Huang, M. Mirakhorli, A. Czauderna, and M. Wieloch, "Decision-Centric Traceability of architectural concerns," in *7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*. San Francisco, CA, USA: IEEE, 2013, pp. 5–11.

[9] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, "How do Practitioners Manage Decision Knowledge during Continuous Software Engineering?" in *31st Int. Conference on Software Engineering and Knowledge Engineering (SEKE'19)*. KSI Research Inc., 2019, pp. 735–740.

[10] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech, "Towards the visualization of usage and decision knowledge in continuous software engineering," in *5th IEEE Working Conference on Software Visualization (VISSOFT 2017)*, Shanghai, China, 2017, pp. 104–108.

[11] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, "Continuous Management of Requirement Decisions Using the ConDec Tools," in *26th International Conference on Requirements Engineering (REFSQ20) Workshops, Doctoral Symposium, Live Studies Track, and Poster Track*. Pisa, Italy: CEUR-WS.org, 2020, p. 6.

[12] A. Kleebaum, B. Paech, J. O. Johanssen, and B. Bruegge, "Continuous Rationale Identification in Issue Tracking and Version Control Systems," in *REFSQ-2021 Workshops, OpenRE, Posters and Tools Track, and Doctoral Symposium*. Essen/Virtual: CEUR-WS.org, 2021, p. 9.

[13] 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.

[14] T.-M. Hesse, V. Lerche, M. Seiler, K. Knoess, and B. Paech, "Documented decision-making strategies and decision knowledge in open source projects: An empirical study on Firefox issue reports," *Information and Software Technology*, vol. 79, pp. 36–51, 2016.

[15] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, "Teaching rationale management in agile project courses," in *16. Workshop Software Engineering im Unterricht der Hochschulen (SEUH)*, 2019, pp. 125–132.

[16] B. Paech, A. Delater, and T.-M. Hesse, "Supporting Project Management Through Integrated Management of System and Project Knowledge," in *Software Project Management in a Changing World*. Heidelberg, Germany: Springer, 2014, ch. 7, pp. 157–192.

[17] T.-M. Hesse, "Supporting software development by an integrated documentation model for decisions," PhD thesis, Heidelberg University, 2020.

[18] P. Hübner and B. Paech, "Interaction-based creation and maintenance of continuously usable trace links between requirements and source code," *Empirical Software Engineering (ESE)*, vol. 25, no. 5, pp. 4350–4377, 2020.

[19] J. S. van der Ven and J. Bosch, "Making the right decision: Supporting architects with design decision data," in *European Conference on Software Architecture (ECSA)*. Berlin, Heidelberg: Springer, 2013, vol. 7957 of LNCS, pp. 176–183.

[20] U. van Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions," *Journal of Systems and Software*, vol. 85, no. 4, pp. 795–820, 2012.

[21] S. Lehnert, "A taxonomy for software change impact analysis," in *12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution (IWPSE-EVOL'11)*. Szeged, Hungary: ACM, 2011, p. 41–50.

[22] A. Caraban, E. Karapanos, D. Gonçalves, and P. Campos, "23 ways to nudge: A review of technology-mediated nudging in human-computer interaction," in *Conference on Human Factors in Computing Systems (CHI)*. Glasgow, UK: ACM, 2019, pp. 1–15.

[23] J. E. Burge and D. C. Brown, "SEURAT: Integrated Rationale Management," in *International Conference on Software Engineering (ICSE)*. Leipzig: IEEE, 2008, pp. 835–838.

[24] J. Malloy and J. Burge, "SEURAT_Edu: A Tool to Assist and Assess Student Decision-Making in Design," in *47th ACM Technical Symposium on Computing Science Education (SIGCSE'16)*. Memphis, Tennessee, USA: ACM, 2016, pp. 669–674.

[25] 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, 2006.

[26] R. Capilla, F. Nava, and C. Carrillo, "Effort Estimation in Capturing Architectural Knowledge," in *23rd IEEE/ACM International Conference on Automated Software Engineering*. L'Aquila, Italy: IEEE, 2008, pp. 208–217.

[27] C. Manteuffel, D. Tofan, P. Avgeriou, H. Koziolek, and T. Goldschmidt, "Decision architect–A decision documentation tool for industry," *Journal of Systems and Software*, vol. 112, pp. 181–198, 2015.

[28] L. Lee and P. Kruchten, "A tool to visualize architectural design decisions," in *Quality of Software Architectures. Models and Architectures: 4th International Conference on the Quality of Software-Architectures*. Springer, 2008, pp. 43–54.

[29] R. C. de Boer, P. Lago, A. Telea, and H. V. Vliet, "Ontology-Driven Visualization of Architectural Design Decisions," in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*. Cambridge, UK: IEEE, 2009, pp. 51–60.

[30] M. Shahin, P. Liang, and M. R. Khayyambashi, "Improving understandability of architecture design through visualization of architectural design decision," in *Workshop on Sharing and Reusing Architectural Knowledge (SHARK)*. Cape Town, South Africa: ACM, 2010, pp. 88–95.

[31] M. Shahin, P. Liang, and Z. Li, "Architectural design decision visualization for architecture design: Preliminary results of a controlled experiment," in *5th European Conference on Software Architecture (ECSA): Companion Vol.* Essen, Germany: ACM, 2011, pp. 1–8.

[32] G. C. Filho and A. Zisman, "D3TraceView: A Traceability Visualization Tool," in *29th International Conference on Software Engineering and Knowledge Engineering (SEKE'17)*. Pittsburgh, PA, USA: KSI Research Inc., 2017, pp. 590–595.

[33] I. Bacher, B. M. Namee, and J. D. Kelleher, "On using tree visualisation techniques to support source code comprehension," in *2016 IEEE Working Conference on Software Visualization (VISSOFT)*. Raleigh, NC, USA: IEEE, 2016, pp. 91–95.

[34] S. Kugele and D. Antkowiak, "Visualization of Trace Links and Change Impact Analysis," in *24th International Requirements Engineering Conference Workshops (REW)*. Beijing, China: IEEE, 2016, pp. 165–169.

[35] C. Yang, P. Liang, and P. Avgeriou, "Integrating Agile Practices into Architectural Assumption Management: An Industrial Survey," in *Evaluation and Assessment on Software Engineering (EASE)*. Copenhagen, Denmark: ACM, 2019, pp. 156–165.

[36] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of software architecture design derived from five industrial approaches," *Journal of Systems and Software*, vol. 80, no. 1, pp. 106–126, 2007.

[37] T. Dingsøyr and H. van Vliet, "Introduction to software architecture and knowledge management," in *Software Architecture Knowledge Management*. Berlin, Heidelberg: Springer, 2009, ch. 1, pp. 1–17.

[38] J. F. Hoorn, R. Farenhorst, P. Lago, and H. van Vliet, "The lonesome architect," *Journal of Systems and Software*, vol. 84, no. 9, pp. 1424–1435, 2011.

[39] O. Zimmermann, L. Wegmann, H. Koziolek, and T. Goldschmidt, "Architectural Decision Guidance across Projects: Problem Space Modeling, Decision Backlog Management and Cloud Computing Knowledge," in *12th Working IEEE/IFIP Conference on Software Architecture (WICSA '15)*. Montréal, Québec, Canada: IEEE, 2015, pp. 85–94.