Ontology-Driven Visualization of Architectural Design Decisions

Remco C. de Boer[1], Patricia Lago[1], Alexandru Telea[2], and Hans van Vliet[1]
[1]Department of Computer Science, VU University Amsterdam, the Netherlands
[2]Institute for Mathematics and Computer Science,University of Groningen, the Netherlands
[1]{remco,patricia,hans}@cs.vu.nl, [2]a.c.telea@rug.nl

Abstract

There is a gradual increase of interest to use ontologies to capture architectural knowledge, in particular architectural design decisions. While ontologies seem a viable approach to codification, the application of such codified knowledge to everyday practice may be non-trivial. In particular, browsing and searching an architectural knowledge repository for effective reuse can be cumbersome.

In this paper, we present how ontology-driven visualization of architectural design decisions can be used to assist software product audits, in which independent auditors perform an assessment of a product's quality. Our visualization combines the simplicity of tabular information representation with the power of on-the-fly ontological inference of decision attributes typically used by auditors. In this way, we are able to support the auditors in effectively reusing their know-how, and to actively assist the core aspects of their decision making process, namely trade-off analysis, impact analysis, and if-then scenarios. We demonstrate our visualization with examples from a real-world application.

1 Introduction

The software architecture community is showing a gradual increase of interest to use ontologies to capture architectural knowledge, in particular architectural design decisions. Recent results in this area include work by Kruchten [12], Akerman and Tyree [2], Prabhakar c.s. [3], and Zimmermann et al. [18]. While ontologies seem a viable approach to capture architectural knowledge, applying such codified knowledge to everyday practice may be non-trivial. In particular, it can be hard to explore and search an architectural knowledge repository so that previously captured knowledge can be reused.

In their early work on visualizing codified architectural design decisions [13], Lee and Kruchten distinguished four visualization types: a simple decision *table* showing design decisions with their attributes and a separate table of relations between decisions; a decision *structure* visualization showing decisions and their relations as nodes and edges in a graph; a decision *chronology* visualization showing how decisions evolve; and a decision *impact* visualization that shows which decisions may be impacted by a change. They conclude that more case studies, and also additional visualization techniques, may be required.

Decision tables are the most often used type of visualization for browsing [13]. Yet, such a view has several drawbacks. Most notably, a list or table is not very effective in showing relationships. As such, it ignores much of the added value of using an ontology. On the other hand, a decision-structure visualization, which seems to be the most natural visualization for a decision ontology, has drawbacks too. While it accurately represents decisions and their relationships, the resulting graph can become cluttered and thus incomprehensible for all but the smallest data sets. Moreover, a graph-like visualization strongly deviates from the tabular view that many practitioners are most accustomed to.

In this paper, we present an ontology-driven visualization that combines the strengths of the decision-table and decision-structure visualizations, and overcomes their drawbacks. Our solution revolves around two tabular views that show design decisions and their mutual relations, respectively. While not unlike Lee and Kruchten's decision/relationship tables, our visualization adds to the standard tabular view the capability to infer on-the-fly several decision attributes (*i.e.* columns) from the structural information captured by the decision ontology. Moreover, we provide several interaction and visual highlighting mechanisms that simplify the process of decision-making in the potentially large and complex state space implied by the ontology. Overall, our aim is to enable users to employ the added value of a decision ontology without losing the simplicity of tabular information visualization.

The remainder of this paper is organized as follows. Section 2 introduces the audit context used to demonstrate our visualization. Section 3 briefly overviews QuOnt, an ontology that supports the reuse of architectural knowledge, *i.e.*

quality criteria, in the early stage of software product audits. Section 4 introduces our ontology-driven visualization tool that supports dynamic exploration of ontology-driven audit scenarios. Section 5 outlines the design details behind our tool. Finally, Section 6 concludes this paper.

2 Architectural Knowledge in Audits

Over the past four years, we have collaborated with DNV-CIBIT, a Dutch SME that acts as an independent audit organization in software product quality assessments. One goal of this collaboration was to investigate the role architectural knowledge plays in software product audits, and to improve upon current practices to manage such knowledge.

In a software product audit, a customer asks an independent third party – the audit organization – to assess the quality of a supplier's product. Auditors need to elicit the customer's idea of 'quality' and compare it with actual characteristics of the supplier's software product. Hence, one of the first stages of a software product audit, is for the auditor to translate the customer's idea of quality – often expressed in terms of quality attributes such as "the product should be scalable" or "security is important" – to quality criteria: concrete measures that should, or should not, be present in the product. For example, in a system where security is important, proper user authentication is a likely quality criterion; without such authentication the necessary level of security is unlikely to be reached.

Architectural knowledge in software product audits exists on at least two levels. First, there is architectural knowledge that relates to the actual state of the software product. This knowledge originates from the product's supplier, and it can be found in product artifacts such as code and documentation. Secondly, there is architectural knowledge that relates to the desired state of the software product. This knowledge originates from the customer, is enhanced by the auditor, and takes the form of what we call 'quality criteria'. Whereas the supplier's software product represents architectural decisions that *have been* taken, quality criteria represent architectural decisions that *should have been* taken [7].

We should note that quality criteria need not always map to measures that *contribute* to the desired quality level. Instead, when a certain measure is known to *inhibit* the desired quality level, a quality criterion could be that that measure should *not* be present in the software product. Quality criteria may thus be expressed as an explicit rejection of a particular design option.

Whether or not a measure should be present in a software product is often a matter of trade-offs. For example, when user-friendliness is essential, there may arguably be no user authentication measures present; when security is essential, user authentication measures are mandatory. If both security and user-friendliness are important, the appropriateness of authentication measures depends on which of the two has precedence. Such trade-offs imply that deciding on a particular quality criterion – *i.e.* whether or not a particular measure should be present – can be hard.

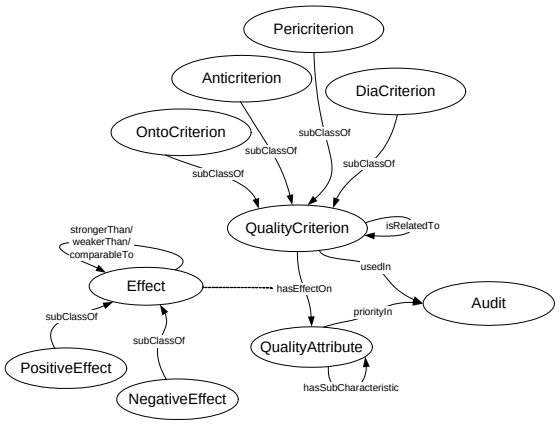


Figure 1. The QuOnt Ontology

The work we present here demonstrates how ontology-driven visual analysis of existing quality criteria can provide decision support for auditors to determine which quality criteria should be used in an audit. This support consists of three main elements: 1. support for trade-off analysis, 2. support for impact analysis, and 3. support for if-then scenarios.

3 QuOnt: An Ontology for the Reuse of Quality Criteria

When performing several software product audits, some quality criteria may be reused. For example, some form of user authentication will be necessary in all high-security systems. In a less-than-ideal setting, such reuse may happen ad-hoc, *e.g.* by re-reading past audit reports to find previously used criteria applicable to the situation at hand. To enable a more structured approach, in previous work [7] we presented the QuOnt ontology that can be used to codify quality criteria for reuse. This ontology forms the basis of our ontology-driven visual analysis discussed here. It consists of the following elements (see Fig. 1):

QualityCriterion is the ontology's main element. The *isRelatedTo* relationships capture how a quality criterion can be related to other quality criteria (discussed in more detail below). We distinguish four types of criteria: ontocriteria (concrete measures or artifacts that must appear in the software product), anticriteria (measures that must not appear), diacriteria (properties that should hold for the whole system and cannot be traced to a single product artifact), and pericriteria (criteria for the audit process itself). For a

Table 1. Relations as constraints on a QuOnt instance

Relation	Constraints	
enables	C1	$\forall x, y : \text{enables}_{x,y} \Rightarrow \neg\text{constrains}_{x,y}$
constrains	C2	$\forall x, y : \text{constrains}_{x,y} \Rightarrow (\neg\text{usedIn}_x \Rightarrow \neg\text{usedIn}_y)$
isBoundTo	C3	$\forall x, y : \text{constrains}_{x,y} \wedge \text{constrains}_{y,x} \Rightarrow \text{isBoundTo}_{x,y}$
	C4	$\forall x, y : \text{isBoundTo}_{x,y} \Rightarrow \text{isBoundTo}_{y,x}$
forbids	C5	$\forall x, y : \text{forbids}_{x,y} \Rightarrow (\text{usedIn}_x \Rightarrow \neg\text{usedIn}_y) \vee \text{overrides}_{y,x}$
subsumes	C6	$\forall x, y : \text{subsumes}_{x,y} \Rightarrow (\text{usedIn}_x \Rightarrow \text{usedIn}_y)$
conflicts	C7	$\forall x, y : \text{conflicts}_{x,y} \Rightarrow \text{forbids}_{x,y} \wedge \text{forbids}_{y,x}$
overrides	C8	$\forall x, y : \text{overrides}_{x,y} \Rightarrow \text{forbids}_{y,x} \wedge \text{usedIn}_x$
alternative	C9	$\forall x, y : \text{alternative}_{x,y} \Rightarrow \neg(\text{usedIn}_x \wedge \text{usedIn}_y)$
	C10	$\forall x, y, z : \text{alternative}_{x,y} \wedge \text{alternative}_{y,z} \Rightarrow \text{alternative}_{x,z}$
comprises	C11	$\forall x, y : \text{comprises}_{x,y_{1,2,\ldots,n}} \Rightarrow (\neg\text{usedIn}_x \Rightarrow \neg\text{usedIn}_{y_1} \wedge \neg\text{usedIn}_{y_2} \wedge \ldots \wedge \neg\text{usedIn}_{y_n})$
depends	C12	$\forall x, y : \text{depends}_{x,y} \Rightarrow \text{constrains}_{y,x} \vee \text{comprises}_{y,x} \vee \text{overrides}_{x,y}$

more detailed discussion of the four criteria types, which are based on the major classes of architectural design decisions from [12], we refer to [7].

QualityAttribute represents a quality attribute that can be further specialized in subattributes. For example, in ISO 9126 'efficiency' is further divided into 'time behaviour', 'resource utilisation', and 'efficiency compliance' [10].

Effect is a reified relation from criterion to quality attribute, having two attributes: effect type (positive or negative) and $[0 \ldots n]$ reciprocal relations to other 'effect' relationships, which indicate the relative strength (stronger than, weaker than, or comparable) of the 'effect' relations.

Audit models a software product audit in which particular quality criteria have been used to assess a prioritized set of quality attributes. The *usedIn* relation captures the relation between criteria and audits. The *priorityIn* relation captures the relation between quality attributes and audits.

Inspired by Kruchten's work in [12], we recognize ten ways in which criteria can be related. These different types of relationships can be expressed as constraints on a QuOnt ontology instance (cf. [7, 9]), as listed in Table 1.

In this paper, we mainly focus on five types of relationships: *constrains*, *subsumes*, *conflicts/forbids*, and *alternative*. First, when a quality criterion X *constrains* another criterion Y, Y cannot be used in the audit unless X is also used (C2). Second, when a quality criterion X *subsumes* another criterion Y, the use of X implies the use of Y (C6). Third, when criterion X *conflicts* with criterion Y, X *forbids* Y and Y *forbids* X (C8); this means that when X is used, Y may not be used and vice versa, unless the decision is made that Y overrides X (C5). Finally, when criterion X is an *alternative* to criterion Y, X and Y cannot both be used at the same time (C9). This is a transitive relation (C10). Of the remaining five relationships, *enables* and *overrides* are not supported by the particular reasoning engine our prototype

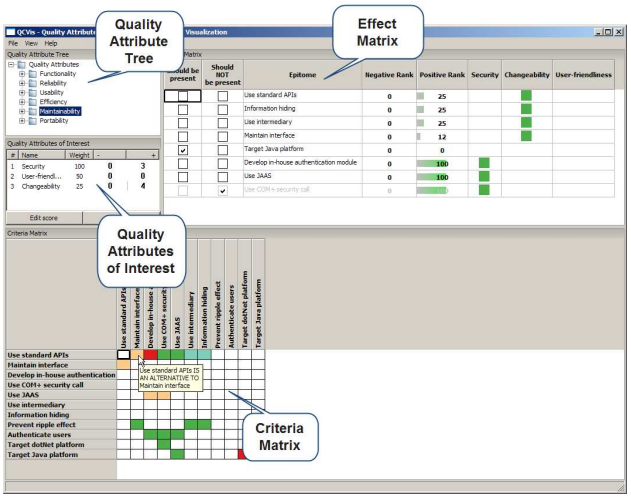


Figure 2. Visualization tool overview

uses (see also Section 5); *isBoundTo* and *depends* are shorthands for combinations of other relations; and *comprises* can be seen as a trivial extension of *constrains*.

4 Ontology-driven Visualization: Three Scenarios

In this section, we shall present our ontology-driven visualization (ODV) tool by means of three usage scenarios derived from an auditor's actual practice, described step-by-step. Although different decision support elements appear in all scenarios, each scenario focuses on a particular aspect of the decision support provided by the ODV; the first scenario focuses on trade-off analysis, the second on impact analysis, and the third on 'if-then' scenarios.

Throughout the scenarios, we shall refer to different widgets, or areas, of our visualization tool, as follows (see also Fig. 2)[1]. The 'Quality attribute tree' shows the hierarchy of quality attributes according to a particular quality model, in this case the extended ISO-9126 or 'Quint' model [17]. The 'Quality attributes of interest' area shows the quality attributes of interest, which capture the customer's idea of 'quality' and are an input to the remainder of the audit. The 'Effect matrix' shows the quality criteria relevant to the current audit. Relevant criteria are those criteria that have a positive or negative effect on one or more attributes of interest, as well as criteria for which it is determined that they should or should not be present in the product. The 'Criteria matrix' shows the relations between quality criteria. All areas are correlated by means of interactive selection and drag-and-drop operations, thereby allowing the auditor

[1] The screenshots in this section are necessarily small, because of space limitations. Larger versions of the screenshots can be obtained from `http://www.cs.vu.nl/~remco/WICSA2009-figures.pdf`

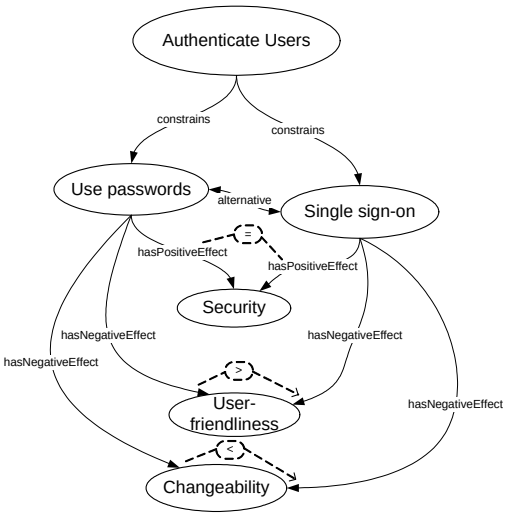


Figure 3. Ontology instance for scenario 1

to both construct and query data to support different audit scenarios, as illustrated next.

4.1 Scenario 1: Trade-off analysis

In the first scenario, an auditor uses the ODV to perform a trade-off analysis for determining which quality criteria to include in an audit. The audit is done on behalf of BSO, a fictional enterprise that wants to assess the quality of a new human resource management (HRM) system being developed by a third-party, which will allow employees to view salary statements and request holiday leave. Together with BSO, the auditor establishes that this HRM system should be secure (first and foremost), user-friendly, but also easily changeable, since BSO's internal IT department will eventually maintain it. Based on this prioritized list of quality attributes (1. security, 2. user-friendliness, 3. changeability), the auditor now needs to determine which quality criteria to use in this audit.

We assume the audit organization has used QuOnt to codify quality criteria from previous audits (not described here). Fig. 3 shows a part of the knowledge base available for this audit. We deliberately consider only part of the knowledge base, so we can focus on just three interrelated quality criteria relevant in our audit: AUTHENTICATE USERS, USE PASSWORDS, and SINGLE SIGN-ON. As visible from Fig. 3, the criteria USE PASSWORDS and SINGLE SIGN-ON are alternatives that are both constrained by AUTHENTICATE USERS. Both of them have a comparable positive effect on security. While the negative effect of the use of passwords on user-friendliness is stronger than that of single-sign on, its negative effect on the product's changeability is weaker.

4.1.1 Quality attribute selection

The auditor's first task is to select and prioritize the quality attributes to be used in his audit. Priorities are quantified on a scale from 1 (lowest) to 100 (highest). The auditor usually elicits these values directly from the customer, *e.g.* using the '100-point method' (or 'hundred-dollar test' [14]) or similar workshop techniques. When such a workshop is infeasible, the auditor may assign a score to the (ordinal) prioritization expressed by the customer, *e.g.* 100 to the highest-priority quality attribute, 90 to the second highest, an so on.

Our ODV tool supports interactive prioritizing of attributes: quality attributes can be dragged from the attribute tree and dropped in the list of quality attributes of interest below. Fig. 4 (step 1) shows this for the 'security' attribute. As part of the drop action, the user is asked to enter the priority score of the selected attribute (Fig. 4, step 2). As soon as a quality attribute has been selected and prioritized, the tool inspects the underlying ontology and updates the effect matrix to display all quality criteria that affect any of the quality attributes of interest. Fig. 4 (step 3) shows the criteria USE PASSWORDS and SINGLE SIGN-ON, in the effect matrix, which both have an effect on the 'security' attribute that was just selected. Just as for 'security', the

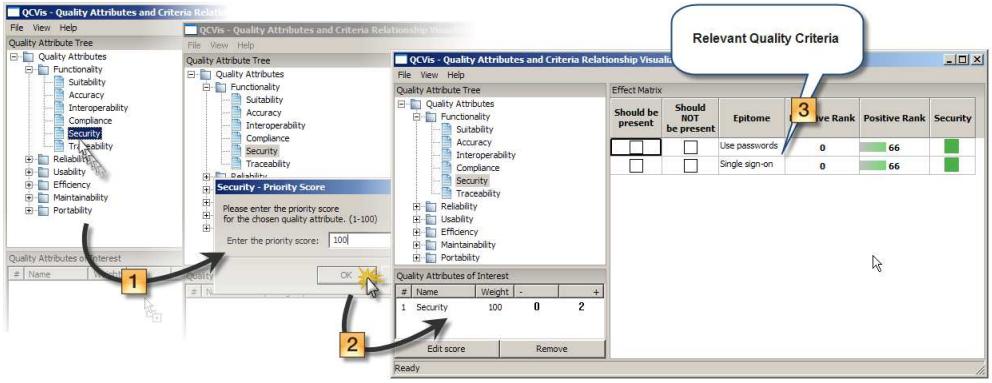


Figure 4. Selection and prioritization of relevant quality attributes for an analysis scenario

auditor selects and prioritizes the two other attributes of interest, 'user-friendliness' and 'changeability', with a score of 50 and 25, respectively. Fig. 5 shows the result.

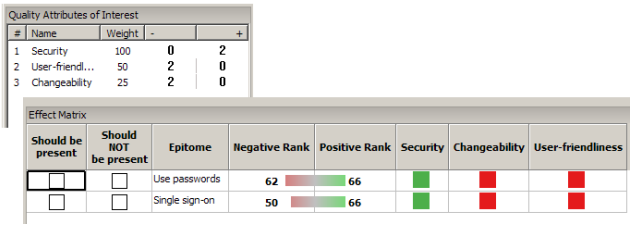


Figure 5. Effect matrix, all attributes of interest

4.1.2 Effect matrix

The effect matrix has a row for each quality criterion relevant to the current audit, and several columns as follows. The rightmost n columns describe each of the n quality attributes of interest. A cell at (row=i,column=j) in these columns is colored red, green, or white to show that criterion i has a negative, positive, and respectively no effect on quality attribute j. In Fig. 5, there are $n = 3$ such columns (the rightmost ones) for our three attributes of interest: security, changeability, and user-friendliness. We see that both criteria have a positive effect on security, and a negative effect on changeability and user-friendliness.

The effect matrix has two additional columns: 'negative rank' and 'positive rank' (columns 4 and 5 in Fig. 5). These show the overall negative, respectively positive effects of a quality criterion on all quality attributes, using scaled color bars. Longer bars represent higher values (also shown numerically[2]). Negative-rank bars are shaded from transparent gray (low values) to saturated red (high values). Positive-rank bars are shaded from transparent gray (low values) to saturated green (high values). In this way, the user's attention is strongly drawn to high positive or negative values, whereas low values are less prominent [15]. In our example in Fig. 5 we see that, although the overall positive effect of both criteria USE PASSWORDS and SINGLE SIGN-ON is comparable, the overall negative effect of USE PASSWORDS exceeds that of SINGLE SIGN-ON (longer bar in column 4, row 2 than in column 4, row 1). This shows that, while both quality criteria have a comparable positive effect on security, USE PASSWORDS has a larger negative effect on user-friendliness (the second-highest priority quality attribute) than SINGLE SIGN-ON.

[2]We calculate these values using partial ordering [5]. The score of a quality attribute is divided by the average rank of a criterion in the set of criteria partially ordered on effect strength. In Fig. 5, the negative value 62 for USE PASSWORDS, for example, is given by $\frac{50}{1} + \frac{25}{2}$, since the average rank of USE PASSWORDS based on the strength of its effect on user-friendliness is 1, and for its effect on changeability 2.

Having seen this, the auditor decides that the SINGLE SIGN-ON criterion should be present in the audited software. This is done by clicking the 'should be present' column checkbox for SINGLE SIGN-ON (row 2, column 1, see Fig. 6). The effect matrix provides also a similar column with checkboxes for 'should not be present' quality criteria. Using these inputs, auditors can indicate which mix of quality criteria best matches the client's requirements.

As soon as the auditor makes his decision by (un)checking a quality criterion, a reasoning engine dynamically inspects the ontology to determine which new facts can be inferred. In this case, since USE PASSWORDS and SINGLE SIGN-ON are alternatives, they cannot be both present in the software product (cf. C9 in Table 1 and Fig. 3). Hence, from the auditor's decision that SINGLE SIGN-ON should be present in the product, the reasoning engine infers that USE PASSWORDS should not be present; the checkbox 'should not be present' of criterion USE PASSWORDS is automatically checked and its name grayed out to reflect this. Moreover, since AUTHENTICATE USERS constrains SINGLE SIGN-ON, the presence of SINGLE SIGN-ON implies that AUTHENTICATE USERS should also be present; so the checkbox 'should be present' of criterion AUTHENTICATE USERS is checked accordingly, and the inference engine adds AUTHENTICATE USERS to the effect matrix. This inference is done automatically, and enables the ontological relations to be reflected directly in the effect matrix. The updated matrix showing which measures should be present is shown in Fig. 6. In this image, the auditor can see all quality criteria for this audit: the HRM system 1. should provide user authentication, 2. should provide a single sign-on facility, and 3. should not use passwords.

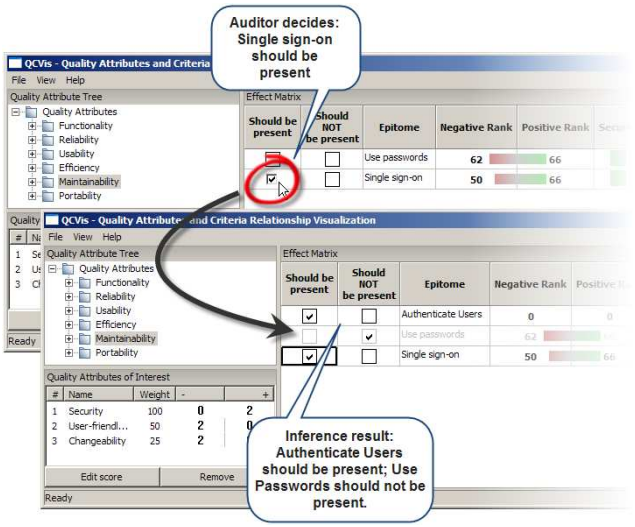


Figure 6. Selection and inference of quality criteria

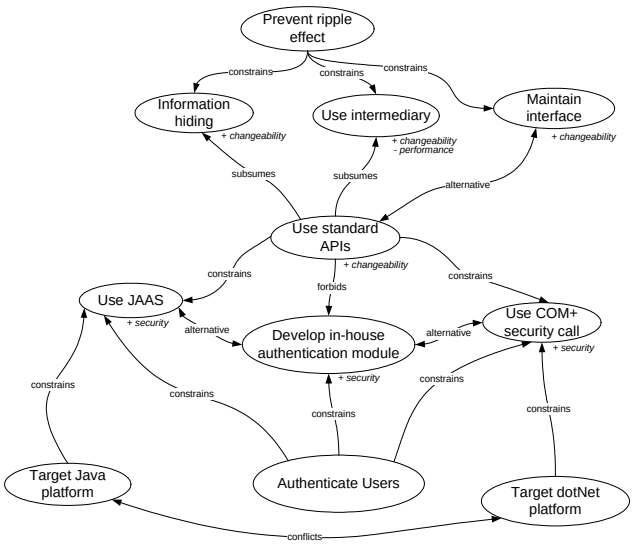


Figure 7. Ontology instance for scenario 2

This simple scenario shows how ODV supports the auditor in deciding which quality criteria to use. The decision input uses the attribute trade-offs, shown as high or low positive or negative ranks. Typical decision-making will have measures with a high positive rank present in the software product, and avoid measures with a high negative rank. When the user records his decision, the reasoning engine determines and visualizes the impact of that decision on other quality criteria. This impact can be analyzed for further decision refinement, as described in the next section for a more complicated scenario.

4.2 Scenario 2: Impact analysis

In this scenario, the auditor has the same goal as in scenario 1: assess the quality of an HRM system. This time, however, we will examine a larger part of the knowledge base, and consider more quality criteria than in the first scenario (see Fig. 7)[3]. Again, the customer indicates that for the HRM system security is the most important quality attribute, followed by user-friendliness and changeability. Additionally, the customer stresses that the product should be built in Java, since his internal IT department only has experience with Java maintenance. Because of the up-front requirement that Java should be used, the auditor starts with a preselected criterion TARGET JAVA PLATFORM, even though no quality attributes have been selected yet (Fig. 8).

[3]For presentation conciseness, we assume that the strengths of all effects on a particular quality attribute are comparable. Hence, Fig. 7 only indicates that a criterion has a positive ('+') or negative ('-') effect on an attribute, followed by the attribute's name. Unlike in Fig. 3, we do not draw relations between the effects that indicate their relative strength so the diagram remains readable.

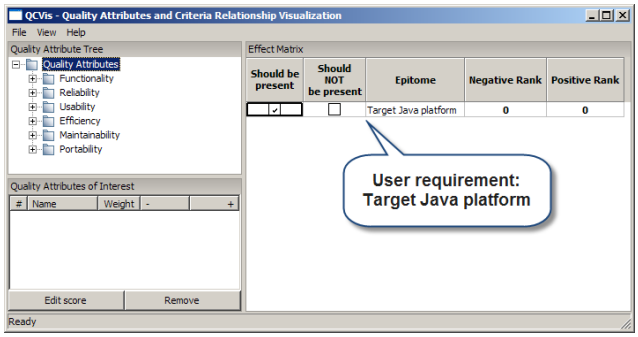
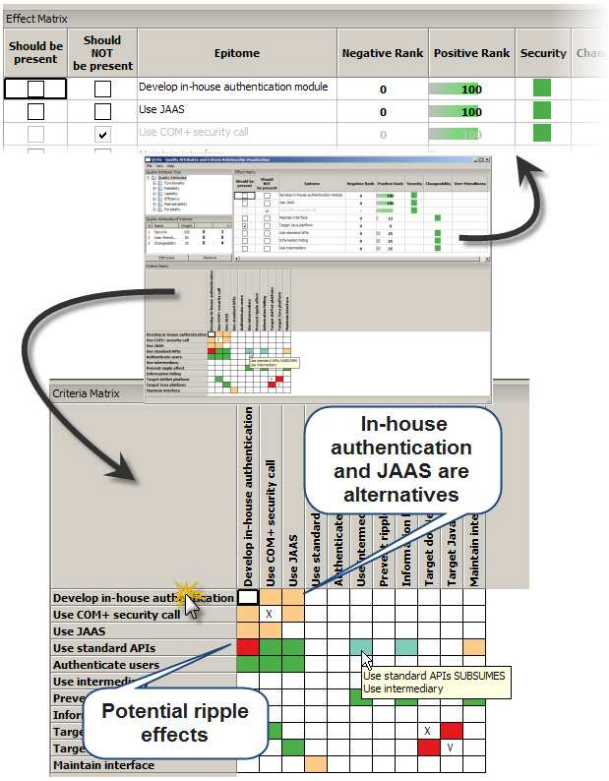
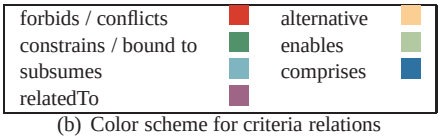


Figure 8. Predefined user requirement

Just as in scenario 1, the auditor selects and assigns priorities to the quality attributes of interest, and views all quality criteria inferred by the tool. With 'security' being the most important quality attribute, the auditor wants to analyze which quality criteria have an effect on security. Hence, he sorts the effect matrix on the 'security' column by clicking the column's label. Now the quality criteria with



(a) Effect matrix and criteria matrix



(b) Color scheme for criteria relations

Figure 9. Security-related analysis

the highest effect on security are placed at the top of the effect matrix, as shown in Fig. 9(a) (top image). These are the topmost three rows of the effect matrix, all having green cells in the 'Security' column; all three effects are positive. Out of these, USE COM+ SECURITY CALL, has been marked 'should not be used' by the reasoning engine, since it is constrained by TARGET DOTNET PLATFORM which in turn conflicts with the initially preselected criterion TARGET JAVA PLATFORM (cf. Fig. 7). This leaves the auditor with two security-related criteria to choose from: USE JAAS and DEVELOP IN-HOUSE AUTHENTICATION MODULE (or DEVELOP IN-HOUSE for short).

4.2.1 Criteria matrix

Since USE JAAS and DEVELOP IN-HOUSE both have a comparable positive effect on security, both are eligible to be selected as practices that should be present in the audited product. To further decide, the auditor wants to inspect the relations between criteria. Our tools supports this task with its third and last area: the Criteria matrix (Fig. 9(a) bottom). This area shows all relations between quality criteria as an adjacency matrix. Each relation, *i.e.* matrix cell, is colored using the color scheme shown in Fig. 9(b). Red cells show relations that, when the auditor decides a particular criterion should be used, inhibit the use of some criteria (*i.e.* 'forbids', 'conflicts', 'alternative'); green cells show relations that imply the use of some criteria ('constrains', 'subsumes'); blue cells show aggregation ('subsumes', 'comprises'); purple denotes generic relations whose nature is not further specified ('relatedTo'). Brushing with the mouse over the matrix cells shows tooltips with details on the relations. For example, the mouse over the cell (USE STANDARD APIS, USE INTERMEDIARY) in Fig. 9(a) shows that "USE STANDARD APIS subsumes USE INTERMEDIARY".

Often, users want to focus on the quality criteria that are most relevant from the perspective of a chosen criterion of interest. We support this as follows. Clicking on a row or column label in the criteria matrix sorts the rows of this matrix so that the criterion of interest, corresponding to the clicked row or column, is placed first (at top) and the criteria that have a direct relation with that criterion are placed immediately thereafter. For example, Fig. 9(a) shows the criteria matrix sorted on the criterion DEVELOP IN-HOUSE.

Using the criteria matrix, the auditor observes that USE JAAS and DEVELOP IN-HOUSE are alternatives, hence only one of them can be present. Moreover, there are some conflicting (red) and enabling (green) relations from USE STANDARD APIS to both criteria. Also, USE STANDARD APIS has relations with other criteria such as INFORMATION HIDING and MAINTAIN INTERFACE, some of which have yet more relations with other criteria, as shown by

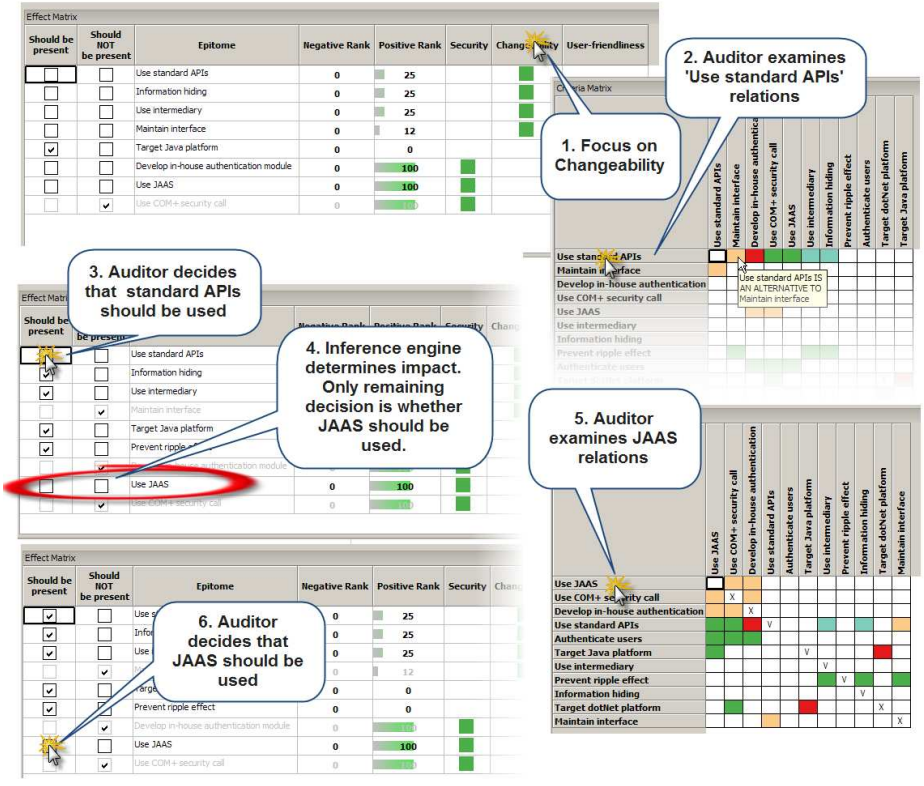


Figure 10. Six-step changeability analysis

the corresponding colored cells in the criteria matrix in Fig. 9(a). This means that a decision on either USE JAAS or DEVELOP IN-HOUSE can have ripple effects on other criteria. Even though the auditor could examine the criteria matrix to trace those effects, it is difficult to see at once which of the two criteria he should expect to be present.

With the security-related alternatives at a tie, the auditor shifts his focus to the next highest-priority quality attribute. Since user-friendliness has neither positive nor negative effects associated with it, the auditor directs his attention to changeability.

Fig. 10 (step 1) shows the effect matrix ordered on the criteria's effect on changeability, *i.e.* with the four changeability-related criteria USE STANDARD APIS, INFORMATION HIDING, USE INTERMEDIARY and MAINTAIN INTERFACE atop. From the criteria matrix in this image, we see (step 2) that USE STANDARD APIS and MAINTAIN INTERFACE (top-left corner of the matrix) are alternatives. Since the former has a higher overall positive rank than the latter (25 vs 12, as shown by the 'positive rank' column in the effect matrix), USE STANDARD APIS is a good candidate to select as a practice required in the software product. Moreover, there are no relations between that criterion and the other changeability-related criteria INFORMATION HIDING and USE INTERMEDIARY that inhibit its selection (*i.e.* no red cells on the intersection of the top matrix row and the 6th and 7th columns). Hence, the auditor decides that USE STANDARD APIS should be present and checks its 'should be used' column accordingly (Fig. 10 step 3).

After the selection of USE STANDARD APIS, the reasoning engine automatically infers which of the other criteria should or should not be present. The result is shown in Fig. 10 step 4. We see that, in addition to USE COM+ SECURITY CALL, criteria MAINTAIN INTERFACE and DEVELOP IN-HOUSE are now also inferred not to be present. From the criteria matrix, the auditor can trace why: USE STANDARD APIS forbids DEVELOP IN-HOUSE, and is an alternative to MAINTAIN INTERFACE. The only choice left for the auditor is whether JAAS should be used or not. Since USE JAAS is an alternative to criteria that we now know should not be present (Fig. 10 step 5), the auditor determines that JAAS should indeed be used (Fig. 10 step 6). This concludes the selection of criteria for this audit.

Summarizing, we have seen how the criteria matrix and the inference engine aid the auditor in determining the impact of his decisions. The criteria matrix provides an overview of the relations between criteria. After the auditor takes a decision that a certain measure should be present or absent, the tool eliminates the need to take decisions that logically follow from that decision, using its inference engine. This saves the auditor time, and can also prevent taking conflicting decisions.

4.3 Scenario 3: If-then scenario

What would have happened in the previous scenario if the auditor had selected a security-related quality criterion instead of considering changeability first? With USE JAAS and DEVELOP IN-HOUSE at a tie, the auditor could choose DEVELOP IN-HOUSE without realizing that this would eventually conflict with USE STANDARD APIS.

In Fig. 11 (step 1), the auditor takes the (as we now know, wrong) decision to expect the presence of an in-house developed authentication module in the software product. Since there are no conflicts yet, the tool accepts the auditor's decision and infers that JAAS should not be used in the product (Fig. 11 step 2). Then, like in the scenario described in Section 4.2, the auditor decides that in this product standard APIs should be used (Fig. 11 step 3).

When the inference engine processes this last decision, it finds a conflict: the use of standard APIs means that an in-house authentication module cannot be used; but the auditor explicitly specified that an in-house authentication module should be used. The tool supports the auditor in detecting and solving such conflicts by marking inconsistent criteria with a red background in the effect matrix (*e.g.* DEVELOP IN-HOUSE in Fig. 11 step 4). The auditor can resolve this conflict manually, by deselecting that DEVELOP IN-HOUSE 'should be used' and setting it to 'should not be used', which creates the opportunity to do the opposite with USE JAAS. Alternatively, the auditor can undo the last steps up to the point where he selected the criterion that is now in an inconsistent state, and continue from there. In that case, the auditor has two remaining options: decide that JAAS should be used or consider another quality attribute first. In both cases, the end result would eventually be the same as the result in Fig. 10 (step 4).

This scenario shows how the auditor can perform if-then scenarios without the need to investigate and trace the intricate set of all relations. From discussions with actual auditors, we know that such scenarios are a valuable decision support mechanism, since they provide immediate feedback on the consequences of (tentative) decisions and thus save time by culling paths in the decision space.

5 Design Rationale of ODV

5.1 Visual Design

We followed several well-known design principles in information- and software visualization [4, 6]. First and foremost, our visual design is simple. Each of our four linked views supports a user task: the quality attribute hierarchy for browsing all available attributes and selecting those of interest; the attributes-of-interest view for selecting attributes for a given analysis; the effect matrix for showing

relations between criteria and attributes and criteria properties; and the criteria matrix for showing relations between criteria (Fig. 2). We use 2D matrix *layouts* to show relations, rather than graphs or 3D layouts. 2D matrix layouts are highly scalable and simple to use, as shown by many software visualization examples [1, 8]. Third, we use a small set of contrasting *colors*, which is effective in attracting the user's attention to salient events, *e.g.* large positive or negative ranks (effect matrix) or conflicting relations (criteria matrix). Finally, *interaction* is simple and directly doable on all views: just a sequence of sorts and selects. Overall, the tool's minimal design and classical GUI made it easily usable and accepted by its target group, and effectively lets users perform 'what if' scenarios in just a few mouse clicks. As such, our tool and its application fits in the newly emerging Visual Analytics discipline [11]: instead of being a static data presentation, our tool guides and supports the user's *decision and reasoning process*. Interaction, linked views, and continuously changing the displayed data based on the decision path are essential elements to this visual analytics design.

The auditors of DNV-CIBIT who assessed our tool reacted very positively. Especially the easy selection of quality criteria and the way the tool invites the user to 'play around' and consider 'what if' scenarios were cited as the tool's main benefits [16].

5.2 Technical Design

The tool's technical design relies on the use of semantic technologies. The ontology is implemented using the 'Web Ontology Language' (OWL), which is endorsed by the World Wide Web consortium and supported by various ontology editors and reasoning engines. The QuOnt ontology presented in Section 3 can be expressed in OWL quite straightforwardly.

The constraints from Table 1 are expressed using the Semantic Web Rule Language (SWRL), an OWL-based rule language. Like OWL, SWRL makes an 'open world' assumption. Briefly, this means that the absence of a statement does not necessarily mean the statement is false. Hence, in the OWL implementation of QuOnt, the absence of the statement that a criterion 'should be used' in an audit does not automatically imply that the criterion 'should not be used'; it only means that it is not known yet whether the criterion should be used. This mimics the way in which auditors reason about quality criteria.

Another implication of the open world assumption is that OWL and SWRL only support monotonic reasoning; only new facts can be introduced and existing facts cannot be changed. Consequently, SWRL does not support negation ($\neg$) nor disjunction ($\vee$). Since many of the constraints in Table 1 use negation and/or disjunction, this poses some problems when modeling QuOnt constraints as SWRL rules.

Fortunately, many of the problems of constraint implementation can be solved. The lack of negation can be largely overcome by introducing a 'notUsedIn' relation for quality criteria that should not be used in an audit. With this new relation, some constraints can be rewritten to eliminate the open world assumption where appropriate. For instance, in SWRL the relation $\text{constrains}_{x,y}$ can be implemented as two rules: $\text{notUsedIn}_x \Rightarrow \text{notUsedIn}_y$ and $\text{usedIn}_y \Rightarrow \text{usedIn}_x$.

Still, two constraints cannot be implemented in SWRL: the 'enables' relation that implies the negation of the 'constrains' relation, which violates monotonic reasoning; and the 'overrides' relation that is a disjunction of the 'forbids' relation. Although this is unfortunate, we found that the 'enables' relation (which is simply defined as 'a weak form of constrains' [12]) has limited practical value. Also, given the open world assumption, the tool still allows to define criteria that at the same time 'should be' and 'should not be' used in an audit, which is in essence the goal of the 'overrides' relation in a closed world assumption.

6 Conclusion

In this paper, we have introduced ontology-driven visualization (ODV) of architectural design decisions, a type of visualization that combines the strengths of tabular and structural visualization and overcomes their drawbacks. We showed how ODV can be employed in a decision support system that assists in the reuse of quality criteria, a particular type of design decision in the early stage of a software product audit. This decision support consists of three main elements: 1. support for trade-off analysis, 2. support for impact analysis, and 3. support for if-then scenarios.

In our work, we have taken the existence of ontology instances such as the ones used in the three scenarios as given. However, one of the largest challenges we see for widespread acceptance of ODV (and for the use of decision ontologies in general) is to overcome the need for complete up-front codification, especially codification of relations of which the potential amount rapidly increases when the number of decisions in the ontology rises.

A particularly interesting codification approach could be an incremental approach, in which ODV plays a role from the very beginning. Since ODV uses whatever information is codified in the ontology, it can be used even on small and/or incomplete knowledge bases. Whenever the user finds some information missing, this information can be added to the ontology and is henceforth available for use and reasoning. In this way, the codified knowledge can be incrementally extended and refined. Moreover, each refinement provides immediate benefit to the user, which provides a strong incentive to improve the knowledge base.

We see a role for data mining techniques that examine the ontology for details on past projects and derive knowledge from this historical data. For example, data mining techniques may label criteria that are often used together as 'relatedTo' each other. When these mined relations are presented to the auditor (*e.g.* in ODV's criteria matrix), the auditor can refine the type of relation in the ontology to one that the inference engine can reason with.

While assessment of our visualization by DNV-CIBIT's auditors was in general positive, we still need more data on the use of ODV in real-life situations. These could be the reuse of quality criteria in audits, but also reuse of design decisions in a forward engineering sense. Further case studies should provide this data to evaluate ODV more extensively.

Acknowledgment

This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge. We thank our master's student Gábor Szabó for his initial work on the visualization. Additionally, we thank the auditors from DNV-CIBIT with whom we have exchanged ideas, in particular Frank Niessink, Mark Hissink Muller, Matthijs Maat, and Viktor Clerc.

References

[1] J. Abello and F. van Ham. Matrix zoom: A visual interface to semi-external graphs. In *Proc. InfoVis*, pages 183–190. IEEE, 2004.

[2] A. Akerman and J. Tyree. Using Ontology to Support Development of Software Architectures. *IBM Systems Journal*, 45(4):813–825, 2006.

[3] L. Babu T., M. Seetha Ramaiah, T. Prabhakar, and D. Rambabu. ArchVoc–Towards an Ontology for Software Architecture . In *Second Workshop on SHAring and Reusing architectural Knowledge / Architecture, Rationale, and Design Intent (SHARK/ADI)*, Minneapolis, MN, USA, 2007.

[4] S. Card, J. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999.

[5] L. Carlsen. Hierarchical Partial Order Ranking. *Environmental Pollution*, 155(2):247–253, 2008.

[6] C. Chen. *Information Visualization - Beyond the Horizon*. Springer, 2004.

[7] R. C. de Boer and H. van Vliet. QuOnt: An Ontology for the Reuse of Quality Criteria. In *Fourth Workshop on SHAring and Reusing architectural Knowledge (SHARK)*, Vancouver, Canada, 2009.

[8] S. Diehl. *Software visualization: Visualizing the structure, behaviour, and evolution of software*. Springer, 2007.

[9] R. Farenhorst and R. C. de Boer. Core Concepts of an Ontology of Architectural Design Decisions. Technical Report IR-IMSE-002, Vrije Universiteit, September 2006.

[10] ISO/IEC. Software engineering - Product quality - Part 1: Quality model. Technical Report ISO/IEC 9126-1, 2001.

[11] D. Keim, G. Andrienko, J.-D. Fekete, C. Goerg, J. Kohlhammer, and G. Melancon. Visual analytics: Definition, process, and challenges. In *Information Visualization - Human-Centered Issues and Perspectives (eds. A. Kerren* et al.*)*, pages 154–175. Springer, 2008.

[12] P. Kruchten. An Ontology of Architectural Design Decisions in Software-Intensive Systems. In *2nd Groningen Workshop on Software Variability Management*, Groningen, NL, 2004.

[13] L. Lee and P. Kruchten. A Tool to Visualize Architectural Design Decisions. In S. Becker, F. Plasil, and R. Reussner, editors, *4th International Conference on the Quality of Software-Architectures (QoSA)*, volume 5281 of *LNCS*, pages 43–54, Karlsruhe, Germany, 2008. Springer.

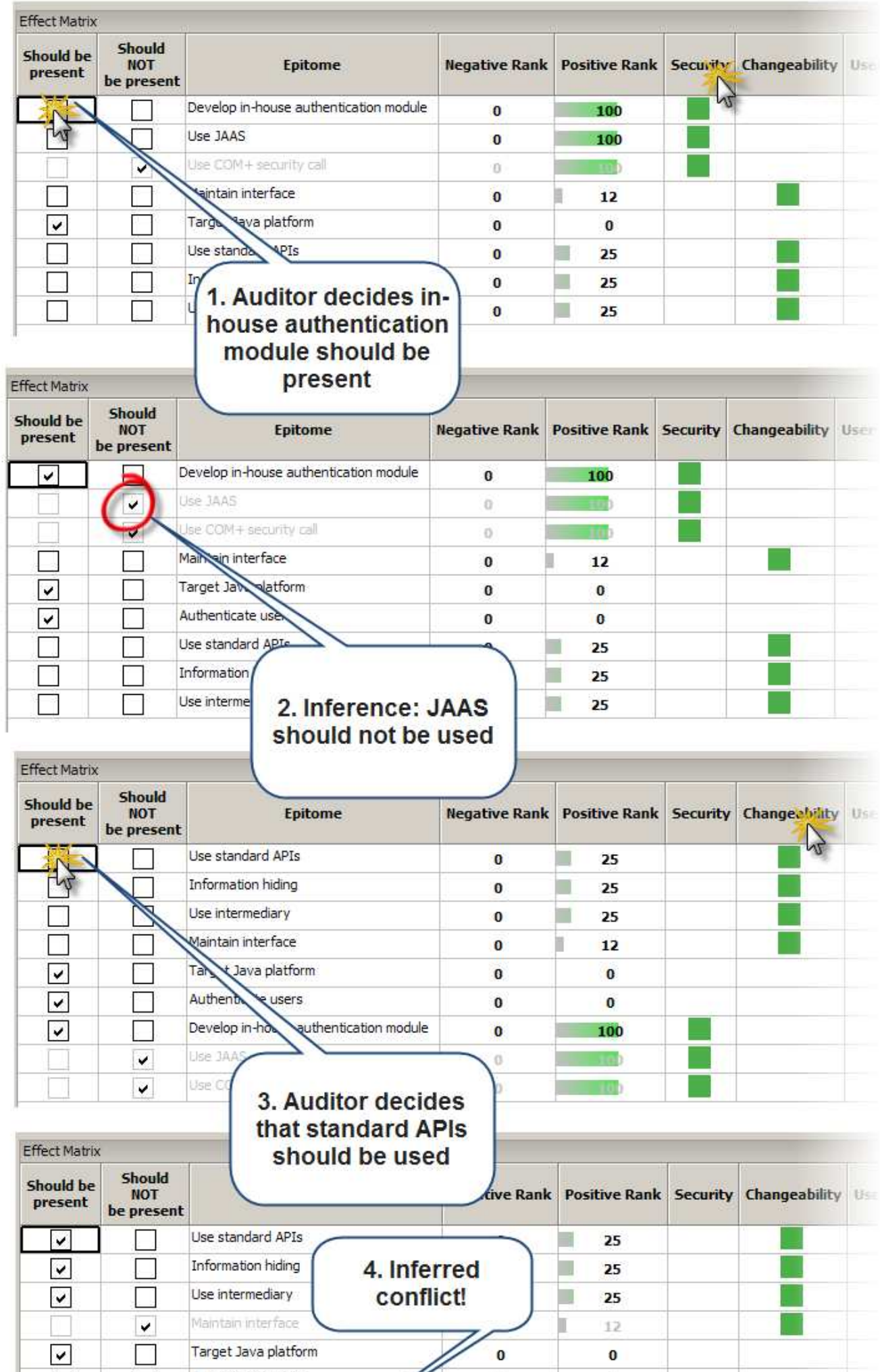
[14] D. Leffingwell and D. Widrig. *Managing Software Requirements: A Use Case Approach*. Pearson Education, 2003.

[15] R. Spence. *Information visualization: Design for interaction*. Prentice Hall, 2nd edition, 2007.

[16] G. Szabó. *Visualization of Complex Quality Criteria – Tool Support for the Software Audit Process*. Master's thesis, VU University Amsterdam, 2008.

[17] B. van Zeist, P. Hendriks, R. Paulussen, and J. Trienekens. Het Extended ISO-Model voor Softwarekwaliteit. In *Kwaliteit van softwareprodukten*, pages 97–156. Kluwer Bedrijfsinformatie B.V., Deventer, 1996.

[18] 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*, 2009, doi:10.1016/j.jss.2009.01.039.


Effect Matrix
Should be present
Should NOT be present
Epitome
Negative Rank
Positive Rank
Security
Changeability
Develop in-house authentication module
Use JAAS
Use COM+ security call
Target Java platform
Use standard APIs
1. Auditor decides in-house authentication module should be present
2. Inference: JAAS should not be used
Use standard APIs
Information hiding
Use intermediary
Maintain interface
Target Java platform
Develop in-house authentication module
3. Auditor decides that standard APIs should be used
4. Inferred conflict!