A Goal-Question-Metrics Model for Configuration Knowledge Bases

Florian Reinfrank, Gerald Ninaus, Bernhard Peischl, Franz Wotawa
Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/II, 8010 Graz, Austria
{firstname.lastname}@ist.tugraz.at

Abstract. Configuration knowledge bases are a well-established technology for describing configurable products like cars, computers, and financial services. Such knowledge bases are characterized by sets of constraints, variables, and domains. Lot of research has been done for testing knowledge bases, finding conflicts, and recommending repair actions.

In contrast, less work has been done in the area of measuring the quality of configuration knowledge bases. Such quality measurements can help knowledge engineers to improve the maintainability, understandability, and functionality of knowledge bases. Based on a literature review we first give an overview of the state-of-the-art in knowledge base metrics. We will extend the current research by using the goal-question-metrics (GQM) approach of the software engineering discipline to find gaps for the characterization of knowledge bases. We will also identify further metrics to complete the model. The results of this paper help knowledge engineers to reduce the effort to develop and maintain configuration knowledge bases.

1 Introduction

Products like cars, computers, and software product lines can be customized according to consumers' preferences. For supporting users to get valid configurations and to support the manufacturing department in companies to get an overview of their production lines, models of their products are necessary [29]. Knowledge bases describe a part of the real world (e.g., the set of valid product configurations for bikes). The implementation of a knowledge base is typically done cooperatively between domain experts and knowledge engineers [5, 42]. Configuration knowledge bases can be represented, for example, as constraint satisfaction problems (CSP [39]).

Configuration knowledge bases (CKB) represent the complete set of valid product configurations. Adding, changing, and removing constraints of such knowledge bases is necessary, because the set of valid product configurations changes over time. Humans in general and knowledge engineers in particular tend to keep efforts related to knowledge acquisition and maintenance as low as possible. Due to cognitive limitations [11, 21, 37] anomalies such as inconsistencies, redundancies, and well-formedness violations are in CKBs. The CKB maintenance task gets even more complicated, if a couple of knowledge engineers has to develop and maintain the knowledge base.

In this paper we describe how to measure the quality of configuration knowledge bases.[1] Therefore, we use the goal question metric approach (GQM). The first step in the GQM approach is to define possible *goals* for the knowledge base, like understandability, maintainability, and functionality. The achievement of the goals will be measured by answers for a set of *questions*. These answers will be aggregated and weighted. After the aggregation and the weightings of answers, the quality of the current version of the configuration knowledge base can be measured in terms of fulfilling the goals. Third, questions will be answered by sets of *metrics*. In this paper we define goals, questions, and metrics and show, how we can measure them. These results will help knowledge engineers in focusing on relevant aspects of the configuration knowledge base to improve the management of a configuration knowledge base, e.g., the efforts for maintainability, understandability, and functionality of a knowledge base.

The remainder of this paper is organized as follows: Section 2 introduces a working example for this work, anomalies and their definitions. Section 3 gives an overview of the state of the art for goals, questions, and metrics for configuration knowledge bases and other research areas. In Section 4 we discuss the GQM model and compare it with the results of other research areas. Finally, Section 5 summarizes this paper and gives an overview of further research in this area.

2 Configuration Knowledge Base

A configuration knowledge base can be defined as a triple (V, D, C) with a set of variables V where each variable $v \in V$ has a domain $dom(v) \in D$. For example, a bike configuration knowledge base could contain the variables $V = \{Reflector, Pedal, Framecolor\}$ where each variable has a domain $D = \{dom(Reflector) = \{yes, no\}, dom(Pedal) = \{Standard,\ Clip\}, dom(Framecolor) = \{Green,\ Red\}\}$. The assignments to a variable (e.g., $Pedal = Clip$) are defined as constraints $c \in C$ [39]. While such assignments

[1] Please consider, that the approach presented in this paper can also be used in other models like knowledge-based recommendation [9] and feature models [4].

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

are defined by users, other constraints are defined by domain experts and are restricting the number of valid combinations of variable assignments. For example, a constraint $c_i = \neg(Pedal = Standard \wedge Framecolor = Red)$ does not allow a consistent configuration with $Pedal = Standard$ and $Framecolor = Red$. We call user assignments *customer requirements* $c \in C_R$ and constraints which are describing the relationship between customer requirements and product variables knowledge constraints $c \in C_{KB}$. The sets C_R and C_{KB} describe C, such that $C_R \cup C_{KB} = C$ [18]. Customers try to find a valid configuration for the configuration knowledge base which means, that they assign values to variables $\{C_R\}$ and the set of assignments is not in conflict with C_{KB} (see Definitions 1 and 2).

Definition1: A **consistent configuration** is defined as a set of customer constraints C_R. This set of constraints is not in conflict with C_{KB}, such that, there exists one possibility to assign a value to each variable, formally defined as $C_{KB} \cup C_R$ is consistent.

Definition2: A configuration is a **consistent complete configuration**, iff the knowledge base is a consistent configuration (see Definition 1) and each variable has an assignment, such that $\nexists_{v_i \in V} v_i = \emptyset$.

Figure 1 gives an example about a configuration knowledge base for a bike domain. The graphic is based on the notation of feature models [6] where each variable is represented as a feature and each feature has the same domain ($dom(v_i) = \{true, false\}$). The notation of constraints of feature models is described in Figure 1.

The following configuration knowledge base (CKB) reflects a constraint-based (CSP-based [39]) representation of the configuration model depicted in Figure 1. Each of the constraints in Figure 1 is part of the set C_{KB}.

$$
\begin{aligned}
&V = \{\\
&\quad Bike, Reflector, Pedal, Framecolor, Green, Red\\
&\quad Standard, Clip\\
&\}\\
&D = \{\\
&\quad dom(Bike) = dom(Reflector) =\\
&\quad dom(Pedal) = dom(Framecolor) =\\
&\quad dom(Green) = dom(Red) = dom(Standard) =\\
&\quad dom(Clip) = \{true, false\}\\
&\}\\
&C_{KB} = \{\\
&\quad c_0 : Bike = true;\\
&\quad c_1 : Bike = true \leftrightarrow Reflector = true;\\
&\quad c_2 : Bike = true \leftrightarrow Pedal = true;\\
&\quad c_3 : Bike = true \leftrightarrow Framecolor = true;\\
&\quad c_4 : Reflector = true \rightarrow Pedal = true;\\
&\quad c_5 : \neg(Pedal = true \wedge Framecolor = true);\\
&\quad c_6 : Framecolor = true \leftrightarrow (Green = true \vee\\
&\qquad Red = true);\\
&\quad c_7 : (Standard = true \leftrightarrow (Clip = false \wedge\\
&\qquad Pedal = true)) \wedge (Clip = true \leftrightarrow (Standard\\
&\qquad = false \wedge Pedal = true));\\
&\}
\end{aligned}
$$

In this example there are different types of anomalies. *Anomalies are patterns in data that do not conform to a well defined notion of normal behavior* [10]. Anomalies can be conflicts, redundancies and forms of well-formedness violations. For a detailed description of anomalies we refer the reader to [14, 16, 20, 28].

In our example, we can see that the set of constraints $\{c_0, c_2, c_3, c_5\}$ is in conflict because there does not exist a valid assignment for these constraints such that all constraints are fulfilled. We call such scenarios conflicts [23] and they are defined in the Definitions 3 and 4.

Definition 3: a **conflict** is given, iff there exists a set of constraints CS which can not result in a valid configuration (see Definitions 1 and 2), such that $CS \subseteq C$ is inconsistent.

Definition 4: a **minimal conflict** is given, if the constraint set CS leads to a conflict (see Definition 3), and there does not exist a subset of CS with the same property of being a conflict, such that, $\nexists CS' \subset CS$ is inconsistent.

The example of this paper contains one minimal conflict: $CS_1 = \{c_0, c_2, c_3, c_5\}$ because this constraint set leads to no valid configuration of the configuration knowledge base and it is not possible to remove a constraint from CS_1 without loosing the property of beeing a minimal conflict (see Definitions 3 and 4). A minimal conflict can be calculated with the QUICKXPLAIN algorithm [23]. If our model has more than one conflict and we want to have all minimal conflicts we need to calculate an acyclic graph (HSDAG) which is defined in [35].

Solutions for such conflicts are called diagnoses [35, 17]. By removing a set of constraints Δ from the configuration knowledge base, we receive at least one valid assignment for each variable of a configuration knowledge base, formally described in the Definitions 5 and 6.

Definition 5: A set of constraints Δ is called **diagnosis**, iff the removal of Δ from the knowledge base C leads to a valid configuration (see Definition 1 and 2), such that $C \setminus \Delta$ is consistent.

Definition 6: A set of constraints Δ is called **minimal diagnosis**, iff it is a diagnoses (see Definition 5) and it is not possible to remove a constraint from Δ without loosing the property of being a diagnoses, such that $C \setminus \Delta' \subset \Delta$ is consistent.

In our example the constraint sets $\Delta_1 = \{c_0\}$, $\Delta_2 = \{c_2\}$, $\Delta_3 = \{c_3\}$ and $\Delta_4 = \{c_5\}$ are minimal diagnosis because removing one of these constraint sets would lead to a consistent configuration knowledge base.

While conflicts and diagnosis are well discussed, little attention has been done to redundancies in configuration knowledge bases. Piette [28] and Felfernig et al. [20] focused on the problem of redundant constraint sets in knowledge bases and defined the term redundancy (see Definition 7).

Definition 7: A set of constraints R is **redundant**, if the removal of R leads to the same semantics of C, such that, $C \setminus R \models R$.

In our example, we have two different sets of redundant constraints: $R_1 = \{c_2\}$ and $R_2 = \{c_4\}$. A redundancy does not have an impact on the semantics of a knowledge base but probably leads to a higher effort for maintenance tasks of knowledge bases and decreases the performance of the configuration knowledge base. We can calculate such sets with the SEQUENTIAL [28] or CoreDiag [20] algorithm. Both algorithms use the negation of C ($\overline{C} = \neg c_0 \vee \neg c_1 \vee ... \vee \neg c_n\}$) for calculating redundant constraints. For checking the semantics of $C \setminus R$ the algorithms check, if $\overline{C \setminus R} \cup C$ is inconsistent. An

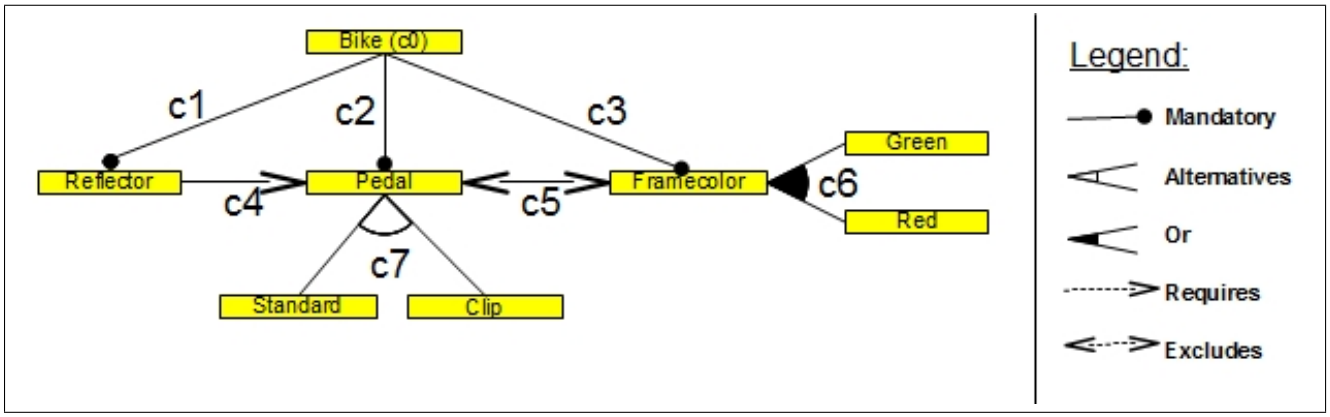

Figure 1. Feature Model of the bike configuration example.

inconsistency means, that the set R is redundant. For getting all different sets of redundant constraints we can also use HSDAG [35].

The third type of anomalies are **well-formedness violations**. Such violations do not have an impact on the consistency of a knowledge base (see Definitions 1 and 2). While well-formedness is well discussed in the research area of feature models (see e.g., [6]), less research has focused on well-formedness violations in configuration knowledge bases. How we use well-formedness violations for calculating metrics for configuration knowledge bases will be described in the next Section.

Anomalies and other aspects of a configuration knowledge base have impacts on the maintainability, understandability, and functionality of the knowledge base. In the following, we give an overview of goals, questions, and metrics for configuration knowledge bases which helps knowledge engineers to get an understanding of the quality of the knowledge base.

While conflicts, diagnoses, and redundancies focus on constraints, well-formedness violations identify anomalies based on variables and domain elements. We now introduce well-formedness violations in configuration knowledge bases.

The first well-formedness violation focuses on dead domain elements. A dead domain element is an element which can never be assigned to its variable in a consistent instance (see Definition 1). Definition 8 introduces a formal description of dead elements.

Definition 8:: A domain element $val_1 \in dom(v_i)$ is **dead** iff it is never part of a consistent instance, s.t. $C_{KB} \cup \{v_i = val_1;\}$ is inconsistent.

On the other hand, we can have domain elements which have to be assigned to each consistent instance. We denote such domain elements *full mandatory* (see Definition 9).

Definition 9:: A domain element $val_1 \in dom(v_i)$ is **full mandatory**, iff there is no consistent (complete or incomplete) instance where the variable v_i does not have the assignment val_1, s.t. $C_{KB} \cup \{v_i \neq val_1\}$ is inconsistent.

The configuration knowledge base can never be consistent, if $Bike = false$ or $Reflector = false$ or $Pedal = false$. For that domain elements, we can say that these domain elements are full mandatory and the other domain element $false$ is a dead domain element. Note that all domain elements of a domain are false optional if one domain element is full mandatory.

Another well-formedness violation is called unnecessary refinement. Such an unnecessary refinement consists of two variables. If the first variable has an assignment, it is possible to predict the assignment of the other variable. A formal definition is given in Definition 10.

Definition 10 (**Unnecessary refinement**): A configuration knowledge base contains a variable pair v_i, v_j. For each domain element val_1 of variable v_i, we can say that variable v_j always has the same assignment $v_j = val_2$, s.t. $\forall_{val_1 \in dom(v_i)} \exists_{val_2 \in dom(v_j)} v_i = val_1 \wedge v_j \neq val_2$ is inconsistent.

In our example the variable pair $Bike$ and $Reflector$ is unnecessary refined because whenever $Bike = true$ the $Reflector = true$, and $Bike = false$ respectively $Reflector = false$ leads to an inconsistency. If such a violation occurs, we can recommend the knowledge engineer to remove the variable $Reflector$ and rename the variable $Bike$ with $BikeWithReflectors$.

3 A GQM model for Configuration Knowledge Bases

For the overview of the metrics for configuration knowledge bases we use the GQM method. For each goal we use a set of questions to define the achievement of each goal. It is also necessary that the goals, questions, and metrics can be calculated automatically, and with explanations [2, 36].

In this section we first give an overview of the possible goals for configuration knowledge bases. Thereafter we give an overview of the questions in (configuration) knowledge bases. Finally we operationalize the questions by listing metrics for configuration knowledge bases.

3.1 Goals for Configuration Knowledge Bases

Nabil et al. [27] define five basic goals for knowledge bases. *Reusability* means, that the knowledge base can be reused in another application area. The *flexibility* defines the possibility to change the semantics of the configuration knowledge base. *Understandability* defines the possibility that knowledge engineers have correct assumptions. *Functionality* describes the applicability of the knowledge base. For example, if the model does not describe the real product assortment, the knowledge

base has no functionality. *Extendability* describes the possibility to extend the knowledge base. In our example (see Figure 1) we can extend the model by adding the bike type ($V' = V \cup \{Type, MountainBike, CityBike\}; D' = D \cup \{dom(Type) = dom(MountainBike) = dom(CityBike) = \{true, false\}\};$).

Lethbridge [25] identifies three goals for knowledge bases. First, it is necessary that knowledge engineers can *monitor* their work. Therefore it is necessary to offer baselines for its continous improvement. Another aspect is the support for knowledge engineers when they *maintain* a knowledge base. Finally, Lethbridge also focuses on the *understandability* of knowledge bases.

From the perspective of software product lines there are three goals: the *analyzability* focuses on the capability of a system to be diagnosed for anomalies. *Changeability* is the possibility and ease of change in a model when modifications are necessary. *Understandability* also means the likelihood that knowledge engineers and designers understand the knowledge base [2].

To sum up, we define the following goals for configuration knowledge bases:

- A configuration knowledge base must be **maintainable**, such that it is easy to change the semantics of the knowledge base in a desired manner [2, 27].
- A configuration knowledge base must be **understandable**, such that the effort for a maintainability task for a knowledge engineer can be evaluated [2, 25, 27].
- A configuration knowledge base must be **functional**, such that it represents a part of the real world (e.g. a bike configuration knowledge base) [27].

3.2 Questions for Configuration Knowledge Bases

After defining the goals for configuration knowledge bases we describe the questions relating to one or more goals. The concordance with the application will be defined by the *completeness*. It suggests the applicability of the current state of the knowledge base for representing the application area. For instance, in our example (see Figure 1) a *Bike* can have a green and a red frame color. If it is possible to combine those colors, the coverage is high. If a frame can only have either a red or a green frame color, the model does not represent the application area and the coverage will be low.

Q1: Is the configuration knowledge base complete?

Anomalies are a well researched area in the context of configuration knowledge bases [30]. The term 'anomalies' is used synonymously for errors and subsumes the terms inconsistencies, redundancies, and well-formedness violations. Errors can have an impact on each of the goals, since it has negative impacts on the reusability, maintainability, and understandability. It can also have a negative impact on the functionality, if there exists an inconsistency in the knowledge base.

Q2: Does the configuration knowledge base contain anomalies?

The *performance* describes the time which is required to calculate characteristics of a knowledge base. These characteristics are e.g., error checking, calculating consistent configurations, and generating user recommendations. This performance mainly influences the functionality of a system (latency) and the reusability.

Q3: Does the configuration knowledge base have an admissible performance?

If it is necessary to develop and maintain the knowledge base a high *modifiability* will help to reduce the effort for the update operation. The modifiability has a positive impact on the reusability and maintainability of a knowledge base. For example, when updating redundant constraints in a knowledge base (e.g. constraint c_2 in the example in Section 2) it's probably necessary to update the redundant constraints (c_4) too. This may lead to a low functionality because the knowledge base doesn't have the correct behavior.

Q4: Is the configuration knowledge base modifiable?

The development effort describes the effort when updating a configuration knowledge base. This effort contains the time for the update operation. This includes the update of the semantics of the knowledge base and the time, which is required to remove all new errors. This effort has an impact on the maintainability and reusability of a knowledge base and is mainly influenced by the *understandability* of a configuration knowledge base.

Q5: Is the configuration knowledge base understandable?

Not each goal has a relationship with each question. In Table 1 we give an overview of the relationship between goals and questions:

Question / Goal	MT	US	FT
Q1 (completeness)			+
Q2 (anomalies)	-	-	-
Q3 (performance)			+
Q4 (modifiability)	+		
Q5 (understandability)		+	

Table 1. Relations between goals and metrics (MT = maintainability, US = usability, FT = functionality)

3.3 Metrics for Configuration Knowledge Bases

The metrics are based on a literature review focusing on knowledge engineering [3, 5, 7, 16, 25, 26, 27, 30, 31, 32, 40] as well as on software product line engineering [2, 6, 22, 24]. The assumptions in this section are based on the literature of configuration knowledge bases and other research areas like feature models and software product lines, software engineering, and rule-based knowledge bases.

After having defined the questions for configuration knowledge bases, the next task is to quantify the metrics. Therefore, we describe possible metrics for configuration knowledge bases. Most of the metrics require a consistent CKB. The metrics are based on literature study in configuration, feature model and software engineering research areas.

The next list shows some metrics derived from MOOSE and function point analysis [2, 12, 25, 36]:

- **Number of variables** $|V|$: In the example (see Figure 1) $|V| = 8$.

- **Average domain size:** $domsize = \frac{\sum_{v_i \in V} |dom(v_i)|}{|V|} = 2$
- **Number of constraints:** $|C| = 8$

The **number of minimal conflicts** $|CS|$ is the first anomaly metric [23]. In our example (see Section 2), we have 1 minimal conflict, such that $|CS| = 1$. We can also evaluate the smallest number of constraints in a conflict set. The lowest number of constraints in a conflict set CS is the **minimal cardinality of conflict sets** $MCCS$ and can be defined as $\nexists CS_j : |CS_j| < |CS_i|$. The example in Section 2 has a minimal cardinality $MCCS = 4$.

We can also evaluate diagnoses for knowledge bases. For example, with the FastDiag algorithm [17, 19] we can calculate the **number of diagnoses** $|\Delta|$ and the number of constraints in a **minimal cardinality diagnosis** $MC\Delta$. A minimal cardinality diagnosis Δ_i is a minimal diagnosis which has the property of having the smallest number of constraints in the diagnosis, such that, $\nexists \Delta_j : |\Delta_j| < |\Delta_i|$. The example described in Section 2, contains 4 minimal diagnoses ($\Delta_1 = \{c_0\}, \Delta_2 = \{c_3\}, \Delta_3 = \{c_5\}, \Delta_4 = \{c_1, c_2\}, |\Delta| = 4$) and a minimal cardinality diagnosis of 1 ($MC\Delta = 1$).

The number of redundant constraints can also be used as a measure for knowledge bases [20, 28, 30, 31] if the configuration knowledge base is consistent.[2] The number of sets of **redundant constraints** is denoted as $|R|$ and the **maximum cardinality of a redundancy set** R_i is 1. We calculate the maximum cardinality for R_i by checking, if there exists another set R_j which has a bigger cardinality, such that R_i has the property of having the maximum cardinality, iff $\nexists R_j \, |R_j| < |R_i|$. The example in Section 2 contains one set with redundant constraints ($R_1 = \{c_4\}, |R| = 1$) and the maximum cardinality of these sets is 1 ($MCR = |1|$).

A domain element $dom_i \in dom(v_j)$ is a **dead domain element**, iff there does not exist a valid configuration, such that, $C \cup \{v_j = dom_i;\} \neq \emptyset$ [5, 6]. When assuming that $C' = C \cup \{c_8 : Standard = true;\}$ it is not possible, that $Clip$ is also $true$, such that, $C' \cup \{c_9 : Pedal = true\} = \emptyset$, such that, $DE = 1$. We use the sum of all dead elements as a metric $0 < DE < 1$ by using Equation 1 where a value nearer 0 means that there are no or less dead elements and a value nearer to 1 means that a high number of domain elements in the knowledge base can not be selected in a consistent configuration knowledge base.

$$DE = \frac{\sum_{v_i \in V} \sum_{d_j \in dom(v_i)} \begin{cases} 0 & C \cup \{v_i = d_j\} \neq \emptyset \\ 1 & else \end{cases}}{|V| \times domsize} \quad (1)$$

A domain element $dom_i \in dom(v_j)$ becomes dead if another domain element $dom_k \in dom(v_l), v_j \neq v_l$ is selected (*Conditionally dead domain elements CD* [6]). In the example (see Section 2) the constraint c_7 does not allow the configuration $Standard = true \wedge Clip = true$.

On the other hand, a domain element can be **full mandatory** (FM). Full mandatory means, that there does not exist a consistent instance of the knowledge base where this domain element isn't selected, formaly described as:

$$FM = \frac{\sum_{v_i \in V} \sum_{d_j \in dom(v_i)} \begin{cases} 0 & C \cup \{v_i \neq d_j\} = \emptyset \\ 1 & else \end{cases}}{|V| \times domsize} \quad (2)$$

Since each domain in our example knowledge base has two values ($true, false$) we can say, that whenever a domain element is dead, the other value becomes full mandatory automatically. When domains have more than two values, it can be the case, that a domain element is dead but there is no other domain value with the property of being a full mandatory domain element.

The third well-formedness violation is called **unnecessary refinement** (UR). Such a violation occurs when there are two variables and the domain element of the first variable in a valid configuration can be suggested by the assignment of a second variable. An unnecessary refinement can be described as $dom(v_i) \rightarrow dom(v_j)$.

In the example in Section 2 we can say that the variables $Standard$ and $Clip$ are an unnecessary refinement, because whenever $Standard = true$ $Clip = false$ and $Standard = false$ $Clip = true$. In that case, we can recommend, that the domain of the variable $Pedal$ can be replaced by $Standard, Clip$ and the variables $Standard$ and $Clip$ can be removed from the knowledge base without changing the semantics of the knowledge base.

The **restriction rate** RR compares the number of constraints with the number of variables. In the example described in Section 2 the restriction rate $RR = \frac{|C|}{|V|} = \frac{8}{8} = 1$. A value greater than 1 means that there is a high restriction [2, 25].

The metric RR is influenced by the design of the knowledge base. For example, while one knowledge engineer requires a single constraint for subsuming the constraints $c_0 \wedge c_1 \wedge c_2 \wedge c_3$ another knowledge engineer is using four single constraints. To consider these different design approaches in the metric, the **restriction rate** RR_2 is considering the number of variables in a constraint, such that, $RR_2 = \frac{\sum_{c_i \in C} \frac{\#vars(c_i)}{\#vars(C)} |C|}{|V|}$ where $\#vars(c_i)$ is the number of variables in c_i.

Another metric from the domain of software engineering is the **variable inheritance factor** VIF [1]. Adapted for configuration knowledge bases, we define VIF as the number of constraints in which a variable v_i appears related to the number of constraints, e.g., $VIF(Framecolor) = \frac{\sum_{c_i \in C} \begin{cases} 1 & v_{framecolor} \in c_i \\ 0 & else \end{cases}}{|C|} = 0.375$ because the variable $framecolor$ appears in three constraints and $|C| = 8$.

To receive a CKB metric we calculate the VIF_{all} for all variables. When calculating the arithmetic mean of the VIF_{all} of all variables, we can evaluate the importance distribution of all variables. A value near to 0 means, that all variables have the same importance and should be considered in the same way. On the other hand, a high value means that there are some important and less important variables in the knowledge base. In such cases, it makes sense to focus on the important variables when maintaining the knowledge base. $VIF_{all} = \sum_{v_i \in V} \frac{\sqrt{(VIF(v_i) - \frac{\sum_{v_j \in V} VIF(v_j)}{|V|})^2}}{|V|}$

Finally, we evaluate the metric **coverage**. The *coverage*

[2] To receive a consistent configuration knowledge base we remove the constraint c_5 from the set C.

measures the number of all consistent complete configurations (see Section 2) compared to the maximum number of complete configurations in a knowledge base. In our example in Section 2 the maximum number of configurations is $\prod_{i=0}^{|V|} |dom(v_i)| = 256$ (8 variables and each variable has a domain size 2). This will be compared with the number of consistent configurations. In our example we have the following consistent configurations:

$$
\begin{aligned}
&\{\\
&\quad Bike = true \wedge\\
&\quad Reflector = true \wedge\\
&\quad Pedal = true \wedge\\
&\quad Framecolor = true \wedge\\
&\quad (Standard = false \wedge Clip = true \wedge Green = true \wedge\\
&\qquad Red = false) \vee\\
&\quad (Standard = false \wedge Clip = true \wedge Green = false \wedge\\
&\qquad Red = true) \vee\\
&\quad (Standard = false \wedge Clip = true \wedge Green = true \wedge\\
&\qquad Red = true) \vee\\
&\quad (Standard = true \wedge Clip = false \wedge Green = true \wedge\\
&\qquad Red = false) \vee\\
&\quad (Standard = true \wedge Clip = false \wedge Green = false \wedge\\
&\qquad Red = true) \vee\\
&\quad (Standard = true \wedge Clip = false \wedge Green = true \wedge\\
&\qquad Red = true)\\
&\}
\end{aligned}
$$

Now we can compare the number of consistent configurations (= 6) with the number of all configurations (= 256). This leads to a coverage of $6/256 * 100 = 2.34375\%$ which is very low and the example configuration knowledge base is very restrictive. For the example knowledge base it is quite easy to evaluate all possible combinations of variables and domain elements. For knowledge bases with more variables, domain elements, and constraints we have millions and more possible combinations of variable assignments. For such scenarios we introduced the **simulation technique** in the context of knowledge based systems to approximate the *coverage*. For a detailed description to approximate this metric in large configuration knowledge bases we refer the reader to [33].

Finally, we can refer the questions to the metrics. Table 2 gives an overview of the relationships between the questions and the metrics.

	Q1	Q2	Q3	Q4	Q5
$\|V\|$	+		-		
$domsize$	+		-		
$\|C\|$	+		-		
$\|CS\|$	-	-		-	
$\|\Delta\|$	-	-		-	
$MCCS$					+
$MC\Delta$					+
$\|R\|$		-	-	-	
MCR					+
DE		-	-	-	-
FM		-	-	-	-
UR		-	-	-	-
RR				-	-
RR_2				-	-
VIF_{all}				-	-
$Coverage$				-	-

Table 2. Relations between metrics (rows) and questions (columns). A '-' means, that the metric has a negative impact on the question, '+' represents a positive impact.

For a detailed description of the calculation of metrics focusing on anomalies (conflicts, redundancies, and well-formedness violations) and the *coverage* metric we refer the reader to [33].

4 Discussion

In this Section we want to discuss relevant aspects of several metrics and give an insight in the implementation of the goal-question-metrics in the iCone interface.

Most of the research in the area of configuration knowledge engineering focuses on the area of verifying configuration knowledge bases (**functionality** goal, see Section 3) and ignores the question how to validate the knowledge base [30] (**maintainabilty** and **understandabilty**). Felfernig et al. [15] present an empirical study about the understandability of constraints in knowledge bases but there does not exist a metric for the understandability of constraints and the knowledge base.

Briand et al. [8] measured the effects of the structural complexity of software and its relationship to the **maintainability** of software. Bagheri and Gasevic [2] transferred this model into the area of feature models and found out, that the number of leaf features, the cyclomatic complexity, the flexibility of configuration, and the number of valid configurations influence the maintainability of feature models. While the simple metrics are easy to transfer into configuration knowledge bases, the depth of a tree or the number of valid configurations can not be calculated.

The number of **redundant constraints** is an important metric since a low number of redundant constraints can improve the maintenance task, simplify the understandability, and reduce the time for calculating valid configurations. An important issue in that case is, that redundant constraints can also improve the understandability of a configuration knowledge base. If a redundant constraint is declared as a desired redundant constraint, the metric should not contain such constraints, but should list it as a desired redundancy.

In a simple configuration knowledge base like the example in Section 2 it is easy to calculate the consistency of each possible configuration for the *coverage* metric. For example, in a configuration knowledge base with a medium number of variables (e.g. 10) and average domain size (e.g. 5) we have approximately $10M$ possible configurations. Since it is not possible to calculate so many possible configurations in realtime, we developed a simulation strategy to approximate the number of consistent configurations. For a detailed description of the simulation strategy, we refer the reader to [34].

While showing the GQM to knowledge engineers can help understand and maintain the configuration knowledge base, it is also important to interpret the results. Therefore we implemented a history for each metric in our iCone-interface[3]. When updates in a configuration knowledge base in the iCone-system are saved, a new version of the knowledge base will be created and metrics will be actualized. In Figure 2 we can see the changes of the value of the $DEAD$ elements metric.

[3] iCone is an 'intelligent environment for the development and maintenance of configuration knowledge bases' (`http://ase-projects-studies.ist.tugraz.at:8080/iCone/index.jsp`).

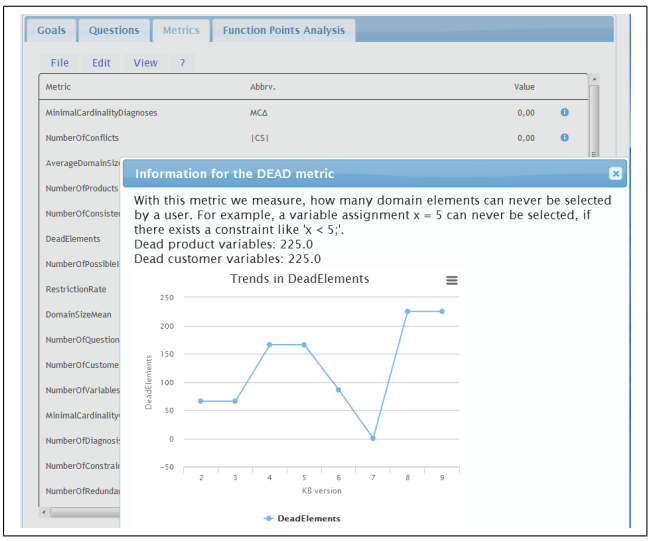

Figure 2. Visualization of changes for the metric *DEAD*. The y-axis shows the number of *DEAD* variables in each version of the configuration knowledge base (x-axis).

Felfernig [13] gives an overview of the usage of **function point analysis** for configuration knowledge bases. Therefore he analyzed the input of a configuration knowledge base from customers and the complexity of a configuration knowledge base. They use the customer requirements as external input (EI), the data, which are required by the user as external query (EQ), the consistent domain elements of variables as external output (EO), knowledge elements as internal logical files (ILF), and external information like the product assortment from an ERP-system as external interface file (EIF). While this approach takes input and output into account, it does not evaluate the quality (e.g., the number of *dead* domain elements) of the input and output.

We have implemented the GQM and the FPA approach in our iCone implementation [41]. Table 3 gives an overview of the performance. Note, that the time contains the calculation / approximation of the metrics and the calculation of all anomalies. The notebook domain is calculated six times and the mobile phone domain is calculated seven times.

	Notebooks	Mobile phones
Product variants	115.00	13,999.00
Product variables	28.00	34.00
product variable domain sizes	1.00 - 45.00	2.00 - 47.00
Customer variables	4.00	5.00
avg. customer variable dom. size	3.75	4.00
constraints	12.00	8.00
min. calc. time	669 msec.	6,811 msec.
max. calc. time	1,715 msec.	18,643 msec.
median calc. time	1,213 msec.	10,842 msec.
mean calc. time	1,252 msec.	11,307 msec.

Table 3. Duration for the calculation of all anomalies (conflicts, diagnoses, redundancies, well-formedness violations), metrics, goal-question-metrics and function-point-analysis for two configuration knowledge bases (notebooks and mobile phones)

In this paper we gave an overview of metrics in configuration knowledge bases, focusing on **knowledge base engineering processes** [38] is out of scope of this paper. We can measure the metrics of an existing configuration knowledge base, but we can not identify the causes of bad configuration knowledge base engineering. To give recommendations for optimizing the knowledge base engineering process, we have to observe the whole process.

5 Conclusion

This paper introduces a goal-question-metric approach to evaluate configuration knowledge bases. We gave an overview of configuration knowledge bases and introduced a running example for this paper and defined goals, questions, and metrics for configuration knowledge bases. Furthermore, we showed how to calculate time-consuming metrics efficiently and presented the iCone-visualization of metrics. It also points out some practical issues when dealing with metrics.

Future research should evaluate the relations between goals, questions, and metrics. Our future work will contain empirical evaluations about the correlation between the goals, questions, and metrics and their weightings in the aggregation process from metrics to questions and from questions to goals.

Future work should take a look at the knowledge engineering process. When calculating metrics for knowledge engineers, we can list some possible improvements. Preece gives an overview about verification and validation techniques for knowledge bases [30]. He aligns different V&V techniques to different models of a knowledge base, e.g., conceptual and design models and an implemented system. The metrics listed in Section 3 only focus on the implemented system. A GQM for the conceptual and design model does not exist.

Another relevant fact is the volatility of requirements [36]. If the requirements for the configuration knowledge base are changing frequently, it is also hard to keep the CKB up to date. For calculating metrics referring to the quality of a CKB and its requirements, it is necessary to integrate a requirements management system in the CKB maintenance tool or offer an interface for both tools.

Acknowledgements

The work presented in this paper has been conducted within the scope of the research project ICONE (Intelligent Assistance for Configuration Knowledge Base Development and Maintenance) funded by the Austrian Research Promotion Agency (827587).

REFERENCES

[1] F. B. Abreu and W. Melo. Evaluating the impact of object-oriented design on software quality. *Proceedings of the 3rd international software metrics symposium*, pages 90 – 99, 1996.

[2] Ebrahim Bagheri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 19(3):579–612, 2011.

[3] Valerie Barr. Applications of rule-base coverage measures to expert system evaluation. *AAAI*, 1997.

[4] Don Batory, David Benavides, and Antonio Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Commun. ACM*, 49:45–47, December 2006.

[5] Joachim Baumeister, Frank Puppe, and Dietmar Seipel. Refactoring methods for knowledge bases. In *Engineering Knowledge in the age of the Semantic Web: 14th international conference, EKAW, LNAI 3257*, pages 157–171. Springer, 2004.

[6] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35:615–636, September 2010.

[7] Jim Blythe, Jihie Kim, Surya Ramachandran, and Yolanda Gil. An integrated environment for knowledge acquisition. *IUI*, pages 14 – 17, 2001.

[8] L.C. Briand, J. Wust, S.V. Ikonomovski, and H. Lounis. Investigating quality factors in object-oriented designs: an industrial case study. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 345–354, 1999.

[9] Robin Burke. Knowledge-based recommender systems. In *Encyclopedia of library and information systems*, page 2000. Marcel Dekker, 2000.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, July 2009.

[11] Yu-Chen Chen, Rong-An Shang, and Chen-Yu Kao. The effects of information overload on consumers' subjective state towards buying decision in the internet shopping environment. *Electronic Commerce Research and Applications*, 8:48 – 58, 2009.

[12] S. R. Chidamber and C. F. Kemerer. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6):476 – 493, 1994.

[13] Alexander Felfernig. Effort estimation for knowledge-based configuration systems. In Frank Maurer and Günther Ruhe, editors, *SEKE*, pages 148–154, 2004.

[14] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, editors. *Knowledge-based configuration. From research to business cases*, volume 1. Morgan Kaufmann, 2014.

[15] Alexander Felfernig, Monika Mandl, Anton Pum, and Monika Schubert. Empirical knowledge engineering: Cognitive aspects in the development of constraint-based recommenders. In Nicols Garca-Pedrajas, Francisco Herrera, Colin Fyfe, Jos Bentez, and Moonis Ali, editors, *Trends in Applied Intelligent Systems*, volume 6096 of *Lecture Notes in Computer Science*, pages 631–640. Springer Berlin / Heidelberg, 2010.

[16] Alexander Felfernig, Florian Reinfrank, and Gerald Ninaus. Resolving anomalies in configuration knowledge bases. *ISMIS*, 1(1):1 – 10, 2012.

[17] Alexander Felfernig and Monika Schubert. Personalized diagnoses for inconsistent user requirements. *AI EDAM*, 25(2):175–183, 2011.

[18] Alexander Felfernig, Monika Schubert, and Stefan Reiterer. Personalized diagnosis for over-constrained problems. *IJCAI*, pages 1990 – 1996, 2013.

[19] Alexander Felfernig, Monika Schubert, and Christoph Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, 26(1):53–62, 2012.

[20] Alexander Felfernig, Christoph Zehentner, and Paul Blazek. Corediag: Eliminating redundancy in constraint sets. In Martin Sachenbacher, Oskar Dressler, and Michael Hofbaur, editors, *DX 2011. 22nd International Workshop on Principles of Diagnosis*, pages 219 – 224, Murnau, GER, 2010.

[21] Yoav Ganzach and Yaacov Schul. The influence of quantity of information and goal framing on decision. *Acta Psychologica*, 89:23 – 36, 1995.

[22] Herman Hartmann and Tim Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *Proceedings of the 2008 12th International Software Product Line Conference*, pages 12–21, Washington, DC, USA, 2008. IEEE Computer Society.

[23] Ulrich Junker. Quickxplain: preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 167–172. AAAI Press, 2004.

[24] Kim Lauenroth and Klaus Pohl. Towards automated consistency checks of product line requirements specifications. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 373–376, New York, NY, USA, 2007. ACM.

[25] Timothy Lethbridge. Metrics for concept-oriented knowledge bases. *International Journal of Software Engineering and Knowledge Engineering*, 8:16–1, 1998.

[26] Mala Mehrotra, Dimitri Bobrovnikoff, Vinay Chaudhri, and Patrick Hayes. A clustering approach for knowledge base analysis. *American Association for Artificial Intelligence*, 2002.

[27] Doaa Nabil, Abeer El-Korany, and A. Sharaf Eldin. Towards a suite of quality metrics for kadss-domain knowledge. *Expert Systems with Applications*, 35:654 – 660, 2008.

[28] Cédric Piette. Let the solver deal with redundancy. In *Proceedings of the 2008 20th IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, pages 67–73, Washington, DC, USA, 2008. IEEE Computer Society.

[29] Joseph B. Pine, editor. *Mass Customization. The new frontier in business competition*. Harvard Business School, 1992.

[30] Alun Preece. Building the right system right evaluating v&v methods in knowledge engineering, 1998.

[31] Alun D. Preece and Rajjan Shinghal. Foundation and application of knowledge base verification. *International Journal of Intelligent Systems 1994;9(8):683702. Duftschmid, S. Miksch /*, 22:23–41, 1994.

[32] ALUN D. Preece, STPHANE Talbot, and Laurence Vignollet. Evaluation of verification tools for knowledge-based systems. *International Journal of Human-Computer Studies*, 47(5):629 – 658, 1997.

[33] Florian Reinfrank, Gerald Ninaus, and Alexander Felfernig. Intelligent techniques for the maintenance of constraint-based systems. *Configuration Workshop*, 2015.

[34] Florian Reinfrank, Gerald Ninaus, Franz Wotawa, and Alexander Felfernig. Maintaining constraint-based configuration systems: Challenges ahead. *Configuration Workshop*, 2015.

[35] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[36] Pedro F. Salvetto, Milton F. Martinez, Carlos D. Luna, and Javier Segovia. A very early estimation of software development time and effort using neural networks. Workshop de Ingeniera de Software y Base de Datos, 2004.

[37] Cheri Speier. The influence of information presentation formats on complex task decision-making performance. *International Journal of Human-Computer Studies*, 64(11):1115 – 1131, 2006.

[38] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowlege Engineering*, 25:161 – 197, 1998.

[39] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[40] William van Melle, Edward H. Shortliffe, and Bruce G. Buchanan. Emycin: A knowledge engineer's tool for constructing rule-based expert systems. In Bruce G. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems. The Mycin Experiments of the Stanford Heuristic Programming Project*, pages 302–313. Addison-Wesley, 1984.

[41] Franz Wotawa, Florian Reinfrank, Gerald Ninaus, and Alexander Felfernig. icone: intelligent environment for the development and maintenance of configuration knowledge bases. *IJCAI 2015 Joint Workshop on Constraints and Preferences for Configuration and Recommendation*, 2015.

[42] Du Zhang and Doan Nguyen. Prepare: A tool for knowledge base verification. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):983–989, 1994.