Preprint:
Baumeister, J.: Advanced Empirical Testing.
In: Knowledge-Based Systems 24(1) (2011), pp. 83-94

Advanced Empirical Testing

Joachim Baumeister

Artificial Intelligence and Applied Computer Science
University of Würzburg, Germany

Abstract

In today's industrial applications, we see that knowledge systems are successfully implemented. However, critical domains require the elaborate and thoughtful validation of the knowledge bases before the deployment. Empirical testing, also known as regression testing, denotes the most popular validation technique, where predefined test cases are used to simulate and review the correct behavior of the system. In this paper, we motivate that the classic notions of a test case and the corresponding measures are not sufficient in many application scenarios. We present enhanced notions generalizing the standard test case, and we show appropriate extensions of the measures precision and recall, that work on these test case notions. Furthermore, the effective inspection of test runs is important whenever test cases fail. We introduce a novel visualization technique that allows for the effective and intuitive analysis of test cases and test run outcomes. The new visualization is useful for debugging a knowledge base and test case, respectively, but it also provides an intuitive overview of the status of the entire test suite. A case study reports on the (repeated) validation of a medical decision-support system and demonstrates the practical relevance of the presented work.

Key words: regression testing, validation, verification, evaluation, test cases, quality measures, knowledge quality, test visualization

1. Introduction

Today, intelligent decision-support systems are widely used in industry, ranging from smart medical applications, e.g., [24, 25, 30, 38], to the automated support of technical and industrial problem-solving tasks, e.g., [22, 31, 33, 36]. Typ-

Email address: joba@uni-wuerzburg.de (Joachim Baumeister)

ically, such systems process inputs—for instance recorded observations—and derive suitable solutions for the given inputs. Whereas some systems are carrying out the work hidden away from the user (commonly closed-loop systems), many systems present an interactive dialog and expect the user to describe the problem by manually entering inputs. In such an interactive setting, we face varying scenarios posing different requirements on the system. In some scenarios, users are willing to enter all available input data before receiving a solution. Other application contexts, however, require a more flexible approach, where possible solutions should be provided as soon as possible. Here, the quality of the proposed solutions improves with more data entered by the user. That way, the user can stop at any time with the data entry as soon as the provided solution is sufficient for his/her current problem.

In general, the quality of the provided solutions is an important criterion for the success and acceptance of the system. Evaluation research in knowledge engineering contributed with validation and verification methods for knowledge bases. In the literature, the terms evaluation, testing, validation, and verification are not used in a consistent manner. Whereas *evaluation* is often used as the generic term, the adjunct tasks of the methods *validation* and *verification* are sometimes mixed. In this paper, we define the *evaluation* of a knowledge base as the generic term of the following methods:

1. *Validation* is the task that checks, if "the right system is built". As a common subtask, *empirical validation* runs test cases with the system, thus evaluating its quality in an operational environment [40]. That way, it is tested whether the system's input/output behavior matches the users' expectations. Common empirical validation methods are *empirical testing* [39] and *robustness testing* [5, 19].
2. *Verification* tests, if the "system was built right". The system is reviewed with respect to known flaws and defective knowledge. Albeit such flaws can be detected by running test cases, the more common subtask *static verification* [41] is typically applied to test for the absence of logical anomalies like redundancies, inconsistencies, and further deficiencies.

This differentiation is in line with the understanding of today's evaluation research, for example see definitions by Ayel and Laurent [3, p. xvi], Gómez-Pérez [17], and Preece [39]. In the context of this article we restrict our considerations to the validation task. For most knowledge representations there exists a wide range of verification methods, that can be applied in a straight–forward way, for example, see [3, 4, 7, 8, 41, 50].

In the past, the validation of knowledge bases (mainly rule bases) was discussed thoroughly. The work mostly focussed on the manual inspection of the knowledge, on the validation using test cases, and on the generation of suitable test cases for a given (rule) base, for example [20, 28, 39, 50]. We see that the *empirical testing* technique denotes a very important and frequently applied validation method. A survey on the practical use of evaluation methods in knowledge engineering is reported in [52]. In software engineering research, empirical testing is often called *regression testing*. Empirical testing is simple and effective: Previously solved test cases with correct solutions are given to the system as input, and the subsequently derived solutions are compared with the expected solutions, that are included in the test cases. The derivation quality is typically measured by precision/recall.

However, the described properties of many systems—interactivity and anytime-solutions—were primarily not considered by these methods. Consequently, we see a gap between the research and their practical applicability in industrial projects. In this article, we revive empirical testing research: We introduce an extension of the classical test case, and we propose appropriate measures for such test cases. In the following, we propose two extensions of testing measures:

1. The *rated precision/recall* that are able to compare solution states rather than the usual boolean occurrences of solutions, i.e., the default states *derived/not derived*.
2. The *chained precision/recall* that not only take into account the final solutions of a case, but also intermediate solutions derived during a problem-solving process. They are also able to weight intermediate solutions in comparison to the final solutions.

Another problem is the efficient and intuitive inspection of the validation results. In practice, test cases need to be reviewed when they failed during an empirical test run or when they were added as new cases to the test suite. We introduce a novel visualization technique, that combines automated testing methods with the manual inspection task of the domain specialist. Test cases are rendered as tree-like graphs, where failed, new, and already inspected cases are represented differently. When printed on paper, such visualizations are the basis for the effective manual inspection of the test cases and the knowledge base behavior, respectively.

The paper is organized as follows: Section 2 introduces the basic data structures for empirical testing, i.e., test cases and their extensions, and we show appropriate extensions of the quality measures precision and recall, that take into account the characteristics of the extended test cases. An important issue is the

inspection of the results of a test run. In Section 3 we describe visualization methods that help during the inspection of empirical test runs. A case study shows the evaluation of a medical consultation system in Section 4. The paper concludes in Section 5 with a summary and a discussion of the approach.

2. Data Structures and Measures

The process of empirical testing usually operates on a suite of test cases, where each test case contains a collection of findings and a set of solutions that are expected for the given findings. In this section, we elaborate this process by formally defining the notions of findings, solutions, and test case. We enhance the classic definition of a test case by the *rated test case* and the *sequential test case*, and we introduce formal definitions of precision and recall, applicable to the extensions of test cases.

2.1. Data Structures: Findings, Solutions, and Test Cases

A knowledge system typically uses a knowledge base to derive suitable outputs (solutions) for a given set of inputs (findings). In the following, we define the basic elements of a knowledge system to be used in empirical testing in more detail.

Definition 1 (Input)**.** *Let $\mathcal{I}$ be the finite set of observable inputs. For every input $i \in \mathcal{I}$ a value range dom(i) is defined, i.e., dom(i) contains all values v that can be assigned to the input $i \in \mathcal{I}$.*

In interactive dialog systems, inputs specify the set of possible questions, that can be presented to the user. The range of possible answers for an input $i \in \mathcal{I}$ is defined by *dom*(*i*), accordingly. For example, a knowledge base contains *temperature* as an input having the possible values *dom(temperature)* = { *low, normal, high* }.

Definition 2 (Finding)**.** *An assignment $f : i = v$ is called a **finding**, where $i \in \mathcal{I}$ is an input and $v \in dom(i)$ is a value assigned to the input. For a set of observable inputs $\mathcal{I}$ we call $\mathcal{F}_\mathcal{I}$ the corresponding universal set of findings, that is defined by all possible combinations of inputs $i \in \mathcal{I}$ and corresponding values $v \in dom(i)$.*

Referring to the previous example, the assignment *temperature=high* is a valid finding $f \in \mathcal{F}_\mathcal{I}$ of the domain, for instance entered by the user of the system. In clear cases we omit the index in $\mathcal{F}_\mathcal{I}$ and only use $\mathcal{F}$ for short.

Definition 3 (Solution)**.** *Let $\mathcal{S}$ be the universal set of* ***solutions*** *that can be derived by the system. An* ***output*** *of the system is an assignment $s = v$, where $s \in \mathcal{S}$ is a solution and $v \in \{true, false\}$ is a boolean value assignable to s. The boolean value "true" assigned to a solution s represents the positive derivation of this solution. That way, solutions represent the possible outputs of the knowledge system.*

Following the simple example from above, we define the solution *fever*, that is derived with the value "true" for the given finding *temperature=high*.

Empirical testing runs a collection of test cases, where a test case contains a list of findings and the expected solutions for the given findings. Formally, a test case can be defined as follows.

Definition 4 (Test Case)**.** *A* ***test case*** *tc is a tuple storing a list of findings and a set of derived solutions:*

$$tc = ([f_1, \ldots, f_p], \{s_1, \ldots, s_q\}), \tag{1}$$

where $[f_1, \ldots, f_p] \subset \mathcal{F}$ are the observed findings and $\{s_1, \ldots, s_q\} \subseteq \mathcal{S}$ are the positively derived solutions, i.e., solutions for which the value "true" was assigned.

In some applications the order of the findings is relevant, for the case that for example some findings represent temporal values of an input. Since there is no order of the derived solutions in the test case every derived solution is equally important. Often it is beneficial to specify a more refined confirmation state of the particular solutions, for example, some solutions are only derived as possible outputs whereas other solutions are categorically derived as a suitable solution. For this reason we extend the definition of solutions by *rated solutions* and introduce the notion of a *rated test case*.

Definition 5 (Rated Solution)**.** *Let R be the universal set of ratings that are used to (partially) order a set of solutions. Let $\mathcal{RS} = \{(s = r) \mid s \in \mathcal{S} \wedge r \in R\}$ be the universal set of rated solutions, where a* ***rated solution*** *$(s = r) \in \mathcal{RS}$ is a rating $r \in R$ assigned to a solution $s \in \mathcal{S}$.*

The domain of ratings depends on the particular knowledge representation used to build the knowledge base. For example, the universal set of ratings R can be defined as the real values in $[0, 1]$ to represent the probabilities derived by a Bayesian network. Alternatively, symbolic values like $R = \{$undefined, excluded, suggested, established$\}$ can be used to express the qualitative rating of a solution. It is easy to see that for the solutions always having the rating $r = established$ a rated solution collapses to a solution in a standard test case as given in Definition 4. Based on the definition of rated solutions we introduce rated test cases.

Definition 6 (Rated Test Case)**.** *A **rated test case** rtc is a tuple consisting of a list of findings $f_i \in \mathcal{F}$ and a set of rated solutions $\{rs_1, \ldots, rs_q\} \subset \mathcal{RS}$:*

$$rtc = ([f_1, \ldots, f_p], \{rs_1, \ldots, rs_q\}) . \tag{2}$$

We denote the findings of the case by $F(rtc) = [f_1, \ldots, f_p]$ and the set of rated solutions of the case by $RS(rtc) = \{rs_1, \ldots, rs_q\}$.

For a default rating $R = \{\text{true, false}\}$ and by omitting the solutions with the rating "false", the rated test case simplifies to a standard test case as introduced in Definition 4.

Although the use of rated test cases improves the testing possibilities, it is sometimes not sufficient to test the derivation quality of the knowledge base *at the end* of the test case. In fact, it is interesting to also test the derivation state *during* the execution of a test case, especially for knowledge bases implementing an anytime system behavior. In order to enable this type of testing, we partition the test case into a sequence of (partial) test cases, where each partial test case stores its findings entered in the particular phase and the solutions (with ratings) derived so far. More formally, we introduce the notion of a *sequential test case*.

Definition 7 (Sequential Test Case)**.** *A **sequential test case** seq is defined by a list of rated test cases rtc_i*

$$stc = [rtc_1, \ldots, rtc_n] ,$$

where a rated test case $rtc_i = ([f_{i,1}, \ldots, f_{i,p}], \{rs_{i,1}, \ldots, rs_{i,q}\})$ depends on its predecessors rtc_j with $j = 1, \ldots, i-1$, i.e., the ratings of $RS(rtc_i)$ are derived based on the observation of findings $F(rtc_i)$ and the previous findings $F(rtc_1), \ldots, F(rtc_{i-1})$.

We see that a sequential test case partitions a standard test case into distinct rated test cases, where every case rtc_i contains an ordered list of findings $[f_{i,1}, \ldots, f_{i,p}]$ that are supposed to be entered by the user in the given order. Additionally, a rated test case rtc_i stores a set of solutions $\{rs_{i,1}, \ldots, rs_{i,q}\}$ with their corresponding ratings that are expected to be derived by a valid knowledge system based on the findings entered in rtc_i and all findings given in the preceding sequences rtc_j with $j < i$.

It is worth noticing that the order of the finding sequences defined in a sequential test case is explicit and important. Thus, every sequence rtc_i depends on its predecessors, especially with respect to the ratings of the particular solutions. The rating of solutions also depends on findings that were entered in previous case sequences.

In summary, a sequential test case $stc = [rtc_1, \ldots, rtc_n]$ is a generalization of a rated test case; for $n = 1$ a sequential test case contains only one sequence and collapses to a rated test case as introduced in Definition 6.

2.2. Traditional Validation Measures

In the previous paragraphs, we defined the basic data structures to be used for empirical testing—the test case, the rated test case, and the sequential test case. For empirical testing a suite of test cases is selected and the findings of each test case are successively batched into a knowledge system. After running each test case the derived solutions are compared with the expected solutions given in the test case. Here, the quality of the derived and the expected solutions, respectively, is computed by standard measures such as precision and recall. In literature, the measures simply compare the set of positively derived solutions with the set of expected solutions.

Definition 8 (Precision)**.** *For a given test case, let $exp \subseteq \mathcal{S}$ be the set of expected solutions and let $der \subseteq \mathcal{S}$ be the set of derived solutions. Then, the* ***precision*** *of the solutions der and exp is defined as follows:*

$$precision(der, exp) = \begin{cases} |\, der \cap exp \,| / |\, der \,| & \text{if } der \neq \{\}\,, \\ 1 & \text{if } der = exp = \{\}\,, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The precision measures how many of the derived solutions were expected to be derived by the case. Analogously, the recall of a test case is defined.

Definition 9 (Recall)**.** *For a given test case, let $exp \subseteq \mathcal{S}$ be the set of expected solutions and let $der \subseteq \mathcal{S}$ be the set of derived solutions. Then, the* ***recall*** *of the solutions der and exp is defined as follows:*

$$recall(der, exp) = \begin{cases} |der \cap exp| \,/\, |exp| & \text{if } exp \neq \{\}\,, \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

The recall measures how many expected solutions were derived by the knowledge base. Sometimes, it is helpful to provide *one* combined measure, that integrates the results of the precision and recall and thus provides a generalizing overview of the test results. Then, the *F-measure* is a favorable metric.

Definition 10 (F-Measure)**.** *For a given test case, let exp* $\subseteq \mathcal{S}$ *be the set of expected solutions and let der* $\subseteq \mathcal{S}$ *be the set of derived solutions. Then, the* ***F-measure*** *compares the solutions der and exp as follows:*

$$f_\beta(der, exp) = \frac{(\beta^2 + 1) \cdot precision(der, exp) \cdot recall(der, exp)}{\beta^2 \cdot precision(der, exp) + recall(der, exp)} \tag{5}$$

The F-measure uses a single equation to weight the outcomes of precision and recall of the derived solutions. Here, the constant $\beta \in \mathbb{R}_+$ is used to weight the calculated precision in relation to the recall. Often, we use the f_1 measure, where precision and recall are defined to be equally important.

Empirical testing uses the presented measures to compare the expected and derived solutions of a test case. Here, a test case *fails* if the computed precision/recall falls below a given threshold; in most applications the measures must not fall below the maximum threshold value 1.

2.3. Extended Validation Measures

Standard precision and recall measures only compare the positive or negative derivation of a solution in a given case. However, when using the extended notions of a test case—the rated test case and the sequential test case—it is reasonable to integrate the additional information stored in the test cases into a set of generalized measures. For example, a generalized precision should be able to compare different ratings of a solution, but also the intermediate ratings of solutions. In summary, an improved set of measures needs to take the following issues into account:

1. The comparison of rated solutions instead of a boolean intersection of the solution occurrences.
2. The evaluation of the quality of chained case sequences instead of one single test case.

Concerning the first issue we introduce "rated" versions of the precision/recall measures that generalize the standard measures and are applicable to arbitrary solution ratings. We further extend these measures by a "sequentialized" version of the precision/recall in order to handle the second issue.

2.3.1. Rated Precision/Recall

The rated precision and recall take into account that the derivation of solutions in a case can be generalized from a boolean occurrence to an explicit rating of the

solution that expresses its degree of confirmation. Since precision and recall compare all solutions occurring in both sets—the derived solutions and the expected solutions—we first define a special intersection function to retrieve all solutions that are contained in both sets independent of their current rating.

Definition 11 (Intersection of Rated Solutions)**.** *Let $RS_1, RS_2 \subset \mathcal{RS}$ be two sets of rated solutions. The **rated intersection** $\cap(RS_1, RS_2)$ is defined by*

$$\cap(RS_1, RS_2) = \left\{ s \in \mathcal{S} \,\middle|\, (s = r_1) \in RS_1 \ \wedge\ (s = r_2) \in RS_2 \right\}. \tag{6}$$

Given the intersection function $\cap(RS_1, RS_2)$ we are now able to extract all relevant solutions in the cases and to compute their similarity of the particular ratings.

Definition 12 (Rated Precision)**.** *Let $exp_{rs} \subset \mathcal{RS}$ be the set of expected solutions of a rated test case, and let $der_{rs} \subset \mathcal{RS}$ be the set of derived solutions. Then, the **rated precision** is defined as*

$$precision_{rs}(der_{rs}, exp_{rs}) = \begin{cases} prec_{rs}(der_{rs}, exp_{rs}) & \text{if } der_{rs} \neq \{\}, \\ 1 & \text{if } der_{rs} = exp_{rs} = \{\}, \\ 0 & \text{otherwise}, \end{cases} \tag{7}$$

where the $prec_{rs}$ is defined by

$$prec_{rs}(der_{rs}, exp_{rs}) = \frac{\sum_{s \in \cap(der_{rs}, exp_{rs})} rsim(r(s, der_{rs}), r(s, exp_{rs}))}{|\, der_{rs} \,|}. \tag{8}$$

The rated precision compares the ratings of solutions by using the abstract similarity function $rsim : R \times R \to [0, 1]$ for ratings $r \in R$ of a solution s contained in der_{rs} as well as in exp_{rs}. The function $r(s, RS)$ returns the rating of solution s in the rated solution set RS, i.e.,

$$r(s, RS) = \begin{cases} r & \text{for } (s = r) \in RS\ , \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Before applying the definition of the rated precision in an application domain, we need to formulate the function for the rated similarity appropriately. It is important to notice that the similarity function *rsim* always returns values between 0 and 1. If it is not defined appropriately for a specific application domain, then we can simply use the *individual similarity function* $rsim_i$ as the default:

$$rsim_i(r(s, RS_1), r(s, RS_2)) = \begin{cases} 1 & \text{if } r(s, RS_1) = r(s, RS_2), \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

When using the individual similarity function, the rated similarity reduces to a boolean comparison as already known from the standard precision measure (see Equation 3).

Examples (Rated Similarity Functions). In the following, we give an example of a possible rated similarity function that could be used for symbolic ratings with the following domain R = {unclear, excluded, suggested, established}.

$$rsim(r(s, der_{rs}), r(s, exp_{rs})) = \begin{cases} 1 & \text{if } r(s, der_{rs}) = r(s, exp_{rs}), \\ 0.8 & \text{if } r(s, der_{rs}) = \text{suggested} \wedge \\ & \quad r(s, exp_{rs}) = \text{established}, \\ 0.5 & \text{if } r(s, der_{rs}) = \text{established} \wedge \\ & \quad r(s, exp_{rs}) = \text{suggested}, \\ 0 & \text{else.} \end{cases} \tag{11}$$

We can see that the asymmetric similarity function uses the intermediate evaluations 0.8 and 0.5, so that it returns a better similarity value, when the expected rating is better than currently derived.

For applications using a numeric value range to define the rating of solutions, for example Bayesian networks, the following similarity function is appropriate:

$$rsim(r(s, der_{rs}), r(s, exp_{rs})) = 1 - \left| r(s, der_{rs}) - r(s, exp_{rs}) \right| \tag{12}$$

In the context of case-based reasoning the notions of similarity are investigated in more detail, and we refer the reader to [12, 45] for a further discussion.

Analogously to the rated precision we define the rated recall of two sets of rated solutions.

Definition 13 (Rated Recall)**.** *Let $exp_{rs} \subset \mathcal{RS}$ be the expected solutions of a rated test case and let $der_{rs} \subset \mathcal{RS}$ be the collection of derived solutions. Then, the **rated recall** is defined as*

$$recall_{rs}(der_{rs}, exp_{rs}) = \begin{cases} rec_{rs}(der_{rs}, exp_{rs}) & \text{if } exp \neq \{\}, \\ 1 & \text{otherwise,} \end{cases} \tag{13}$$

where rec_{rs} is defined by

$$rec_{rs}(der_{rs}, exp_{rs}) = \frac{\sum_{s \in \cap(der_{rs}, exp_{rs})} rsim(r(s, der_{rs}), r(s, exp_{rs}))}{|\, exp_{rs} \,|}.$$

As already introduced, we use the intersection function $\cap(RS_1, RS_2)$ defined in Equation 6 and the rated similarity function $rsim(\dots)$ as discussed before. For the individual similarity function $rsim_i$ (see Equation 10) the rated recall $recall_{rs}$ is equivalent to the standard recall measure *recall* shown in Definition 9.

2.3.2. Chained Precision/Recall

Based on the extensions of precision/recall introduced above, we further generalize the measures to evaluate a knowledge base using a test suite of sequential test cases.

Definition 14 (Chained and Rated Precision)**.** *Let $stc = [rtc_1, \dots, rtc_n]$ be a sequential test case. Every rated test case rtc_i stores its expected solutions $exp_{i,rs} \subset \mathcal{RS}$ in sequence i of the test case stc. Accordingly, we define $der_{i,rs} \subset \mathcal{RS}$ to be the solutions derived by the knowledge base in sequence i. We define the* ***chained and rated precision*** *for $DER_{rs} = (der_{1,rs}, \dots, der_{n,rs})$ and $EXP_{rs} = (exp_{1,rs}, \dots, exp_{n,rs})$ as follows:*

$$precision_{rs,c}(DER_{rs}, EXP_{rs}) = \frac{\sum_{i=1\dots n} w_p(i) \cdot precision_{rs}(der_{i,rs}, exp_{i,rs})}{\sum_{i=1\dots n} w_p(i)}, \qquad (14)$$

where $w_p : \mathbb{N}_+ \to [0, 1]$ defines the weight of the intermediate solutions for every sequence. The measure $precision_{rs}$ is known from Definition 12.

It is easy to see that for $n = 1$ and $w_p(n) = 1$ the chained and rated precision $precision_{rs,c}$ yields the rated precision $precision_{rs}$ as introduced in Definition 12.

Examples (Weight Functions). The appropriate specification of the weights depends on the particular application domain. We see two typical possibilities to define the weights for the chained and rated precision:

- **Equi-important:** The quality of the derived solutions is equally important for every sequence, i.e., $w_p(i) = 1$ for all $i = 1, \dots, n$.
- **Inverse-annealing:** The quality of the derived solutions becomes more important in later sequences and is most important in the final sequence. Then, we define $w_p(i) = i/n$ for $i = 1, \dots, n$.

The definition of the chained and rated recall is analogous to the definition of the chained and rated precision.

Definition 15 (Chained and Rated Recall)**.** *Let $stc = [rtc_1, \ldots, rtc_n]$ be a sequential test case. Every rated test case rtc_i stores its expected solutions $exp_{i,rs} \subset \mathcal{RS}$ at the sequence i of the test case stc. Accordingly, we define $der_{i,rs} \subset \mathcal{RS}$ to be the solutions derived by the knowledge base in sequence i. We compute the* ***chained and rated recall*** *for $DER_{rs} = (der_{1,rs}, \ldots, der_{n,rs})$ and $EXP_{rs} = (exp_{1,rs}, \ldots, exp_{n,rs})$ as follows:*

$$recall_{rs,c}(DER_{rs}, EXP_{rs}) = \frac{\sum_{i=1\ldots n} w_r(i) \cdot recall_{rs}(der_{i,rs}, exp_{i,rs})}{\sum_{i=1\ldots n} w_r(i)}, \qquad (15)$$

where $w_r : \mathbb{N}_+ \to [0, 1]$ defines the weight of the intermediate solutions in sequence i. The measure $recall_{rs}$ is given in Definition 13.

In the context of the chained and rated recall we are able to specify a distinct weighting function w_r in order to define a different weighting scheme compared to the weighting of the computed precisions. However, often the same weighting function is used for w_p and w_r.

2.4. The Test Suite and Total Measures

In practice, a collection of cases, i.e., a *test suite*, is used for the validation of the knowledge base and all cases are rated with respect to their validity. For a proper validation process, we need to ensure that the test suite is consistent with respect to their included test cases.

Definition 16 (Consistent Test Suite)**.** *A test suite TS is called a* ***consistent test suite****, if and only if there exist no two sequential test cases $stc_i, stc_j \in TS$ with the following conditions:*

1. *The first $(q-1)$ sequences of the cases stc_i and stc_j are identical, i.e., $(rtc_{i,1} = rtc_{j,1}), \ldots, (rtc_{i,q-1} = rtc_{j,q-1})$.*
2. *The findings in sequence q are identical but their solutions differ, i.e., $F(rtc_{i,q}) = F(rtc_{j,q})$ and $RS(rtc_{i,q}) \neq RS(rtc_{j,q})$.*

Thus, for two cases with an identical prefix of findings in the rated test cases we expect to also have the identical rated solutions.

It is worth noticing that a test suite with two identical cases is still consistent. A more comprehensive discussion of the consistency of a test suite in the context of software engineering tests can be found for example in [13]. With a consistent test suite the averaged quality of the knowledge base is then defined by the total precision and the total recall.

Definition 17 (Total Precision and Total Recall)**.** *For a given consistent test suite of sequential test cases* $TS = \{stc_1, \ldots, stc_n\}$ *the expected solutions for each test case are given in* stc_i *with* $EXP_{rs}(stc_i)$*, and the solutions currently derived by the knowledge base for this case are denoted by* $DER_{rs}(stc_i)$*. Then, the* ***total precision*** *of the test suite TS is calculated by*

$$Precision(TS) = \frac{\sum_{c \in TS} precision_{rs,c}(\, DER_{rs}(c), EXP_{rs}(c)\,)}{|\,TS\,|}\,, \tag{16}$$

and the ***total recall*** *of the test suite TS is given by*

$$Recall(TS) = \frac{\sum_{c \in TS} recall_{rs,c}(\, DER_{rs}(c), EXP_{rs}(c)\,)}{|\,TS\,|}\,. \tag{17}$$

We see, that the *total F-measure* can be easily computed by using the total precision and the total recall as defined above. All three measures yield real numbers between 0 and 1, where the result 1 denotes a "fully correct knowledge base", i.e., the derived solutions perfectly match the solutions expected in the test cases of the suite.

2.5. Related Work

Since the beginning of expert systems, the validation of developed systems has always been a lively research topic. Reports on early work can be found for example in [3, 14, 26, 42, 48]; all approaches basically consider only the boolean test of the final solutions for test cases. Furthermore, Djelouah et al. [15] investigate the validation of rule-based systems, but postulate a complete test suite. Also, validity is only tested on a boolean level, and no sequential test knowledge can be represented. Santos and Dinh [46] introduce a validation framework for Bayesian knowledge bases. Since the exact comparison of solutions' probabilities is not reasonable in the general case, the validity of the expected and derived solutions is defined by ordering the solutions' ratings, i.e., a test case is valid if there exists no incorrect solution with a higher probability than a correct solution.

In the context of this paper we focus on the measures precision, recall, and F-measure. For some domains, for example medicine, it is also common to measure the validity of test outcomes by the sensitivity, specificity, and intersection accuracy, for example see [49]. These relate true positive and true negative as well as false positive and false negative outcomes of a test. However, the idea of the proposed extensions of precision and recall can be transferred to these measures in an almost direct manner.

3. Inspection and Visualization of Test Cases

In the previous section, we introduced advanced data structures and measures to evaluate the validity of a test suite. For the practical application of empirical testing we need appropriate tools to inspect the results of the testing phase. Especially with a growing size of the test suite, the use of visualization methods can provide an easier access to the validation results. Visualization methods can aggregate and organize the elements of the knowledge base, so that the manual and semi-automated review of the knowledge becomes easier. In the following, we introduce visualization approaches that are applicable for the intuitive inspection of empirical testing runs.

3.1. Visualization using the Unit Testing Metaphor

In the context of the agile development of software systems, the unit testing metaphor has become a popular and successful method [11]; for example, JUnit [16] is a very popular testing framework for Java development. Here, every single feature of a software program is covered by an *automated test*, i.e., a test that already knows the expected results of the method to test. All tests are collected in a test suite that can be launched by a one-click action. The successful run of all tests is indicated by a green status bar, otherwise this bar is colored in red in order to signalize that at least one test failed. With the metaphor of a colored status bar, the developers immediately can check the overall status of the software.

In the past, this approach was also transferred to the knowledge engineering task, where the validity of a knowledge base was tested using a suite of test cases [9, 10]. Test cases—as presented in this paper—are also automated, since the expected results of a case are also stored as its solutions. Here, every test case covers a specific aspect of the knowledge in the best case. Coverage measures for a knowledge base and the test suite, respectively, have been discussed for example in [4]. Figure 1 shows an example of a unit testing tool in practice. The empirical testing tool of the knowledge development environment d3web.KnowME (http://www.d3web.de) follows the unit metaphor and not only presents the exact values for precision, recall and F-measure, but also visualizes the overall result by a colored bar. The result bar moves like a progress bar during the test run and changes the color from green to red as soon as some test cases of the suite failed, i.e., the total F-measure falls below a given threshold (usually set to 1).

In Figure 1, the test run of a medical knowledge base is shown, where a test suite with 5.045 test cases is applied to the knowledge base. Although the

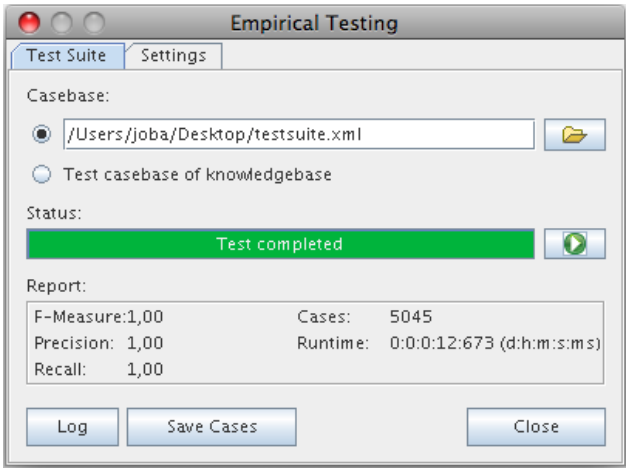

Figure 1: Empirical testing of a knowledge base using the unit testing metaphor with green/red colored bars.

metaphor allows for a quick and intuitive analysis of the overall result, it lacks when errors occur and a deeper analysis becomes important. For detected errors, usually a debugging session of the erroneous test case(s) is initiated, for example see [53] for a discussion of rule base debugging.

Although debugging is a powerful technique to find faulty knowledge, it sometimes fails to efficiently display the context of the erroneous case: That way, often surrounding cases with similar findings have passed the test run successfully and they would help to understand the problem of the faulty case. In addition, common evaluation techniques are not supporting the manual inspection of sequential test cases sufficiently. Here, appropriate visualizations can provide helpful support for the developer of the knowledge base. In the following, we introduce a tree-like visualization method that displays test cases and their contextual surrounding cases in an effective manner.

3.2. Tree-like Visualization with DDTrees

The construction of DDTrees is a novel technique to visualize the cases of a test suite together with the validation results in a compact manner. Furthermore, the visualization can also be used to verify the interactive behavior of a knowledge base, that is represented by the particular sequences of a sequential test case. DDTree stands for *derivation/dialog tree* since it uses a tree structure to visualize the derivation as well as the dialog behavior of the system.

3.2.1. Construction of the DDTree

In summary, a DDTree arranges the cases of a test suite in a tree. Every path from the root of the DDTree to a leaf prints a distinct test case, whereas inner nodes of the tree represent distinct rated test cases that are shared by multiple sequential test cases. More formally, we define a DDTree as a collection of DDNodes and DDEdges. A DDNode contains the information given in a particular sequence rtc_i of a sequential test case $stc \in TS$, i.e., the findings F_i and the rated solutions RS_i of rtc_i.

Definition 18 (DDNode)**.** *Let TS be a consistent test suite and let stc be a sequential test case* $stc = [rtc_1, \ldots, rtc_n]$ *with* $stc \in TS$*. A* ***DDNode*** *n represents at least one sequence* rtc_i *contained in the test cases of TS and is given as the tuple*

$$n = (\, F_n, RS_n, out_n \,),$$

where $F_n = F(rtc_i)$ *is the collection of findings of the sequence and* $RS_n = RS(rtc_i)$ *is the set of rated solutions of the sequence. Furthermore,* $out_n \subseteq \mathcal{I}$ *is a list of inputs, for which values are assigned in the next sequences of the particular test cases. The next sequences of test cases are represented by follow-up nodes in the DDTree.*

Usually, the inputs contained in out_n are presented as subsequent follow-up questions in a dialog with the user. A DDEdge connects two DDNodes, i.e., sequences of a sequential test case, and depicts the findings given in the targeted sequence and DDNode, respectively.

Definition 19 (DDEdge)**.** *A* ***DDEdge*** *e connects two distinct DDNodes* n_i *and* n_j*. For a DDEdge* $e = n_i \rightarrow n_j$ *we call* n_i *the source node of e and* n_j *the target node of edge e. As additional information the findings* F_j *of the target node* n_j *are attached to the edge e.*

The information given on the edges is redundant since the findings are also contained in the target DDNode. However, printing the findings also on the particular edges appears to be intuitive and effective when manually inspecting the resulting tree.

Definition 20 (DDTree)**.** *A* ***DDTree*** *is defined as a poly-tree* $DDT = (N, E)$*, where N is the collection of DDNodes and E is the collection of DDEdges. Let* $TS = \{stc_1, \ldots, stc_n\}$ *be a consistent test suite. For each sequential test case* $stc \in TS$ *a path from the root of the tree to a leaf is defined, and for each maximum*

path in the tree there exists a corresponding test case in TS. For every stc = $[rtc_1, \ldots, rtc_m]$ *in TS the first sequence* rtc_1 *defines a root of the DDTree and the last sequence* rtc_m *is a leaf of the DDTree.*

In summary, the node of a DDTree can belong to a couple of sequential test cases, if and only if this rated test case and the prior sequences of the sequential test cases are identical, i.e., the sequential test cases share the same prefix of rated test cases.

We illustrate the semantics of a DDTree by an example. For a given test suite with the following sequential test cases

$$
\begin{aligned}
TS = \{ & \\
& [([q_1 = a_{1,1}, q_2 = a_{2,1}], \{rs1\}), ([q_3 = a_{3,1}, q_4 = a_{4,2}], \{rs1, rs2\})], \\
& [([q_1 = a_{1,1}, q_2 = a_{2,1}], \{rs1\}), ([q_3 = a_{3,2}, q_4 = a_{4,1}], \{rs1, rs3\})], \\
& [([q_1 = a_{1,2}, q_2 = a_{2,1}], \{rs2\}), ([q_3 = a_{3,1}, q_4 = a_{4,2}], \{rs2, rs4\})] \\
\} &
\end{aligned}
$$

the DDTree is built as depicted in Figure 2. It is important to notice that each path of the DDTree, starting from the root to a leaf, corresponds to a *sequential test case* introduced in Definition 7. Each node of such a path describes a rated test case $rtc_i = ([f_{i,1}, \ldots, f_{i,n}], \{rs_{i,1}, \ldots, rs_{i,q}\})$ of the sequential test case. The questions displayed at the bottom of the node represent possible directions of following sequences. Since the first two test cases of the test suite share the same prefix, they also share the left DDNode at the first level in Figure 2.

We see that a DDTree is able to display the sequences and intermediate solutions of a collection of test cases in a compact manner. The interview and derivation behavior of one single case can be reproduced by navigating from the root to a leaf of the DDTree. Related cases with a similar context can be investigated very easily since often they are printed very close to the given path.

However, for realistic knowledge bases and test suites, respectively, the size of the corresponding DDTree tends to become very large. A large DDTree can be partitioned into smaller trees by extracting the subtrees below the root into single trees that are in turn validated.

3.2.2. Inspecting Test Results with DDTrees

The previous Figure 2 showed sequential test cases in a static way. Figure 3 gives a further example, where also the results of an empirical test run are visual-

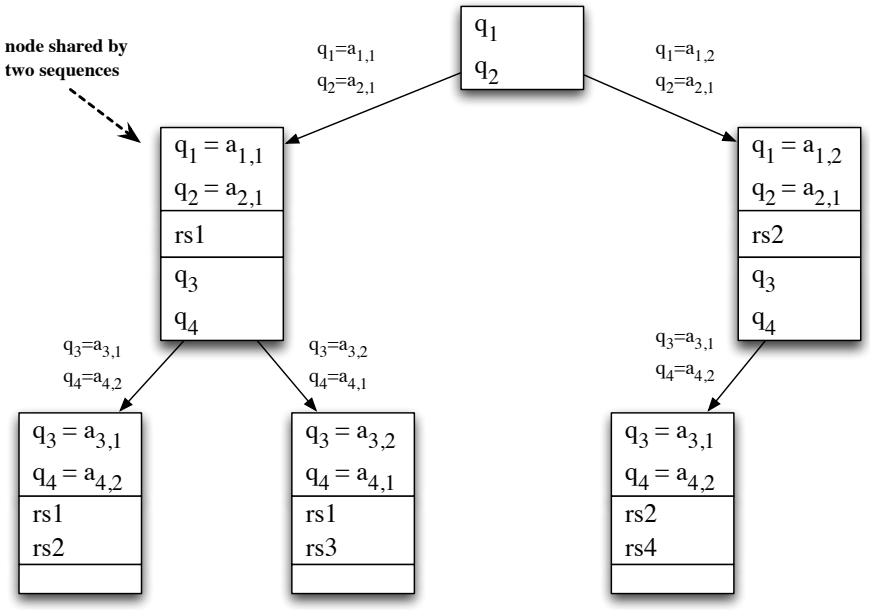

Figure 2: An abstract example of a DDTree generated from three sequential test cases.

ized within the DDTree[1]. The correctness of the derived and expected solutions is rendered by red and green colors of the arcs connecting the tree nodes and the case sequences, respectively. To calculate the correctness, we apply the precision/recall measures introduced in Section 2.3.2. Here, edges of the tree are displayed in green color when this sequential part yielded the expected solutions. Further on, edges of the tree are printed in red color if this edge was part of a faulty sequence, i.e., the F-measure dropped below 1. The example DDTree depicted in Figure 3 contains only cases with one finding for each sequence.

Also, the ratings of the particular solutions are explicitly given as integer points, i.e., score weights corresponding to the confidence of the derivation state. For instance, input *Question 1* is initially asked; for finding *Question 1=yes* the system derives the solutions *Solution 2* and *Solution 3* with 10 points, thereafter *Question 4* is asked. If this input is answered with *yes* then *Solution 2* is rated with 1009 points, whereas *Solution 3* remains at 10 points. In this example the points

[1]In the following figures, the red arcs are also printed in a wider shape in order to identify them more easily in a possible b/w-printed version of this article.

directly correspond to the rating of the solutions.

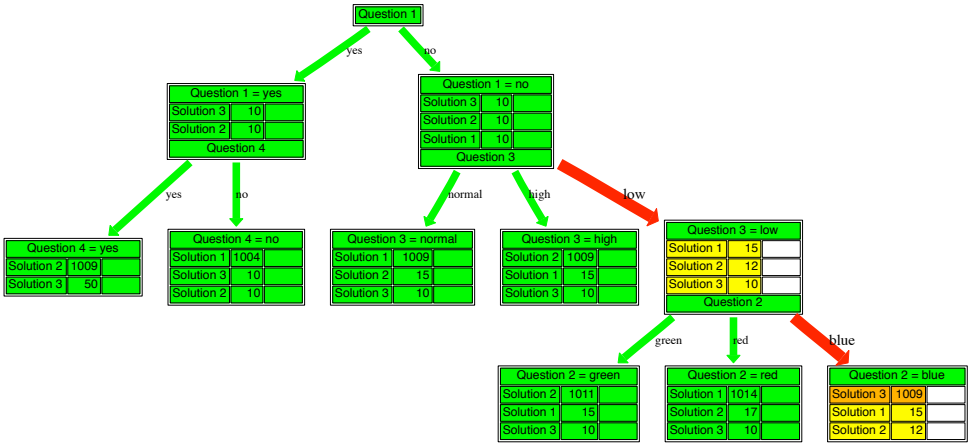

Figure 3: An example DDTree: Each test case is represented by a path from the root to a leaf of the tree. Faulty sequences of cases are colored as red arcs, whereas valid sequences are colored in green.

We see that the DDTree visualization technique generalizes the unit testing metaphor introduced in Section 3.1, since the overall color appearance of the tree visualizes the overall healthiness of the test suite, i.e., the number of correctly solved test cases: The greener the tree appears to the user the "healthier" the knowledge base is. In contrast, a tree displaying many red arcs intuitively signalizes an "unhealthy" knowledge base.

In comparison to the abstract visualization as a status bar of the unit test, the DDTree also shows related cases as a bonus, that are correctly solved in this context. When the developer inspects incorrect cases he/she usually starts with the analysis from the beginning of the incorrect behavior, i.e., the first red arc of the path representing the sequential test case. The preceding and correct beginning of the case is not marked in red color. Thus, the following sequences of the erroneous case are simple to grasp. Since the adjacent and similar cases are also depicted in the tree, the context of the erroneous case is much easier to understand and analyze.

3.2.3. Evolution of the Test Suite and DDTrees

In addition to the inspection of the test suite with respect to correct and faulty test cases, the presented visualization technique allows for the application within

the evolution of the test suite. Often, a test suite grows with the development of the knowledge base. For example, when adding new parts to the knowledge base, also new test cases covering the new features are added to the test suite.

In this context, the DDTree visualization helps to check the new cases in the context of the existing test suite. For the review of new cases it is reasonable to change the color metaphor to improve the analysis: When rendering the extended test suite as a DDTree, we propose to grey out cases that were already contained in the suite before and that were solved correctly in the previous test run. Sequences of newly added test cases are highlighted by drawing the arcs in black color. Thus, the developer can easily identify the new cases and their context, i.e., the relation to related cases in the test suite. Further, faulty cases are always colored in red to signalize the need for a detailed inspection.

Figure 4 shows an example for the coloring of a DDTree to be used during the inspection of an evolved test suite. Already inspected and valid cases are greyed out (left part of the tree), whereas new cases or faulty cases are printed in color so that the inspecting person can focus on the interesting parts of the test suite.

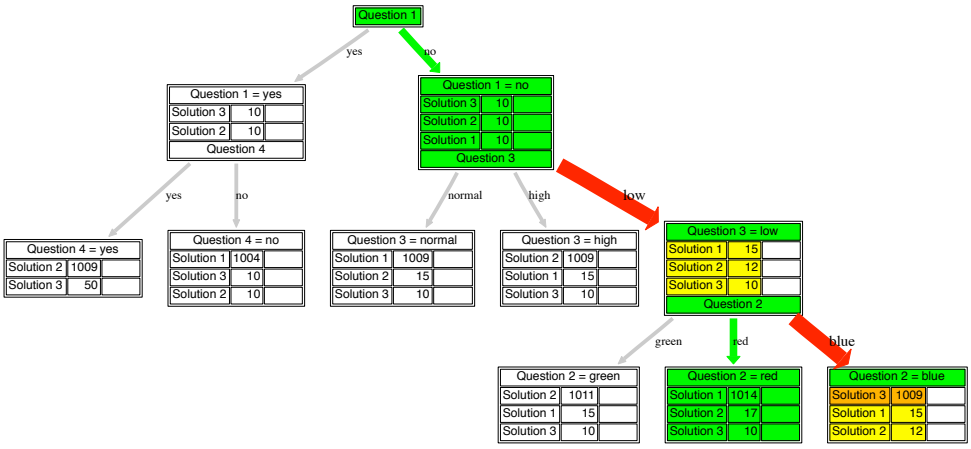

Figure 4: Already inspected and valid test cases are greyed out (left), so that the tester can concentrate on new and faulty tests (right).

For the acknowledgement task of added test cases we propose a manual review process: Each unreviewed case, i.e., every path from the root to a leaf, is manually inspected by a domain specialist (not necessarily the developer of the knowledge base). For this step, we recommend to print out the entire DDTree on a poster (or

the partitioned trees on a couple of posters) in order to obtain a better overview of the interview and derivation workflow. We experienced the classic review on a printed poster as beneficial for a couple of reasons:

1. Usually, domain specialists are not familiar with the handling of (specialized) computer software. Therefore, when using printed paper to display the knowledge base and test suite, respectively, we significantly lower barriers to accomplish the review task.
2. The generated DDTrees are often very large and thus a full display of the entire tree is usually not feasible even on large computer screens. However, having a coherent overview of the test cases is often necessary in order to review a specific case in the context of related (and correct) cases.
3. The review process can be easily documented by highlighting the already inspected sequences with a textmarker pen and by optionally writing comments for refinement instructions directly on the poster. Also, there is no need to learn/cope with specialized software, that often distracts from the review process. As a side-effect, we experienced that visually checking-off accomplished reviews with the textmarker pen increases the motivation of the domain specialist doing the review.

In summary, the presented visualization offers a number of advantages: The domain specialist can easily *see* the context of the current case he/she is inspecting, e.g., what will happen if the question is answered differently, and which solutions are still possible at this stage, etc. Furthermore, no computer skills are required; the specialist can concentrate on the domain knowledge and does not have to possibly struggle with the particular nature of a computer software.

3.3. Related Work

The presented visualization technique is strongly related to *decision trees* [44] and *ordered binary decision diagrams* [2, 23]. Both approaches represent the derivation knowledge in a compact, graph-like manner, that is similar to the described DDTree method. However, the DDTree approach does not aim to display the entire knowledge base and its derivation paths exhaustively, but only the test cases given by a test suite. Moreover, DDTrees additionally use color metaphors to visualize the valid derivation of single test runs.

A more recent approach for a graph-like visualization of derivation knowledge was proposed by *eXtended Tabular Trees* (XTT), for example see [29, 34, 35]. Here, an attribute-set value representation is visualized by a collection of connected derivation graphs. As with decision trees the entire knowledge base is

represented by XTT graphs but not a selection of corresponding test cases. A similar approach was taken by the visualization of *derivation graphs*, see for example [1, 37], that displays the derivation traces for a specific solution [6, 47].

A software engineering-oriented approach was presented in [21], where rules of a reengineered knowledge base are visualized by UML sequence diagrams. The sequence diagrams depict the derivation flow from findings to solutions, and during the inspection of these diagrams test cases are generated. Finally, the test suite consists of all inspected derivation paths.

It is important to notice that all the previous approaches visualize explicit derivation knowledge by graphs, whereas the DDTree approach is independent from the underlying knowledge representation since it displays the derivation knowledge in a black-box manner only using the test cases of the test suite.

4. Case Study

In the context of this work we presented two contributions to the evaluation of intelligent systems: 1) enhanced measures for the precision and recall of derived solutions, 2) a novel visualization method for the interactive inspection of validation results. For both, we report on experiences of their application we have made so far.

4.1. Enhanced Precision and Recall

The presented measures were implemented as a plugin of d3web.KnowME, a visual modeling environment for the development of diagnostic knowledge systems [10]. The newly introduced measures *rated precision* and *recall* are especially useful when validating knowledge bases using an elaborated knowledge representation such as Bayesian networks, heuristic decision trees, and (heuristic) scoring rules. d3web.KnowME supports a variety of (uncertain) knowledge representations to be used for the development of the knowledge base.

Albeit the measures are applicable within arbitrary development projects, they appear to be especially useful in the context of a debugging session. By using sequential test cases, one can define a conditional breakpoint that stops the execution of the test suite once the chained precision and recall fall below a given threshold. Figure 5 shows a screenshot of the debugger of d3web.KnowME, where an example knowledge base for car fault diagnosis is inspected using a test case.

The breakpoints are ordered by their object's type: solutions ("Diagnoses"), input findings ("Questions") and applied derivation knowledge ("Rules"); in this

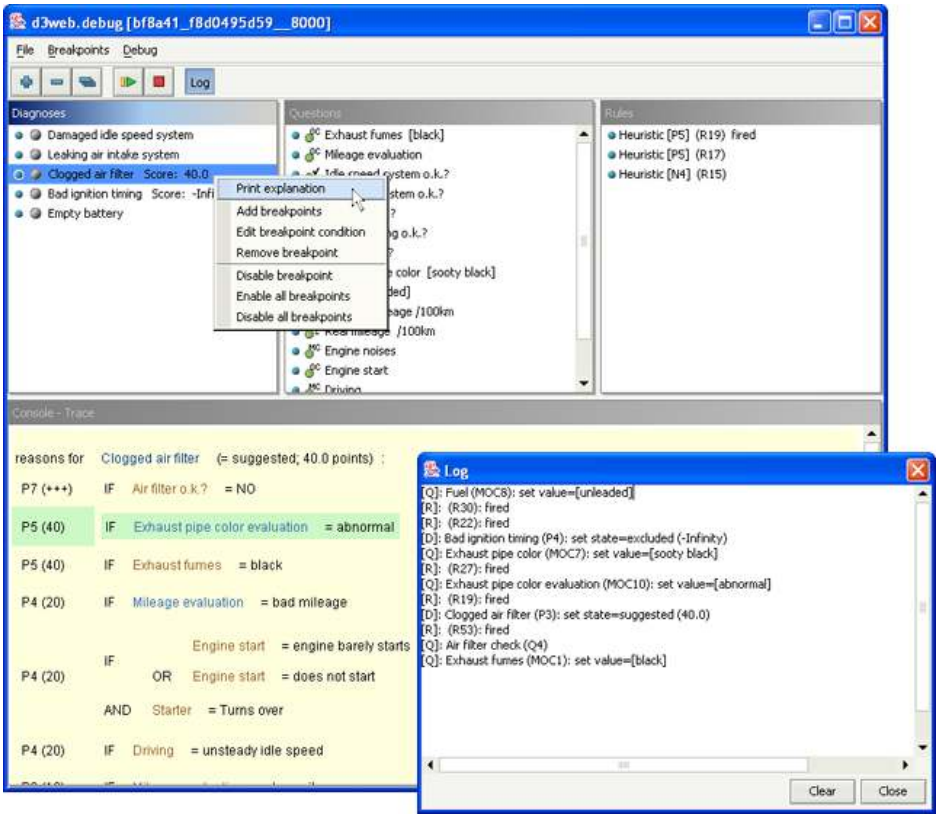

Figure 5: The visual debugger of d3web.KnowME. The top panels display the defined breakpoints for solutions, questions, and rules. At the bottom, the stack trace of the working rule base is depicted.

example, heuristic scoring rules are used to derive the particular solutions. A trace of executed rules is shown in the lower part of the window.

4.2. Visualization by DDTrees

As described earlier, the enhanced measures *chained precision* and *recall* are used for DDTrees to render the tree. The visualization technique was successfully applied in the context of several evaluation phases during the development and evolution of the medical decision-support system *Digitalys CareMate*. The system is commercially sold as part of an equipment kit for medical rescue trucks. It is used as a consultation system during medical rescue services, when the problem definition of a particular rescue mission is complex and a second opinion becomes important.

Currently, the underlying knowledge base comprises about 200 findings that are used to derive a rated scoring of 120 solutions. Besides the rated derivation of suitable solutions, the implementation of an efficient interview technique was the major goal of the project. Thus, the user can be guided through an interview focussing on relevant questions of the current problem definition. With more questions answered, the current ranking of possible solutions improves in relevance, and the interview strategy targets at the presentation of reasonable follow-up questions. The interview strategy follows official school guidelines for emergency medical technicians. An extended version of the knowledge formalization pattern *heuristic decision tree* [43] was used for the implementation of the knowledge base, defining about 1.500 rules for the construction of the interview behavior as well as the rated derivation of the solutions.

Figure 6: A picture taken during the second review phase of the CareMate knowledge base (July 2008). DDTrees are printed on posters and checked using a textmarker pen.

A first evaluation phase was conducted in March 2008 with a prototype of parts of the knowledge base. Test cases were generated automatically from the existing knowledge base by an exhaustive traversal of all possible interview trails. Based on this test suite the DDTree was built, printed on paper, and reviewed in a one-day workshop by the expert panel consisting of two domain specialists. Due to the intuitive semantics of the tree the domain specialists were able to immediately grasp the behavior of the system by manually traversing through the particular tree paths. For each case, alternative case trails could be easily inspected on the paper,

and the discussion about the cases was not interrupted by external circumstances like handling specific computer software.

Before its deployment in version 1.0, the knowledge base was finally reviewed in July 2008. Here, already checked test cases as well as newly generated test cases were printed on a collection of 11 posters comprising the entire test suite. The test suite consisted of 2.051 test cases; the generated cases were created by exhaustively traversing all possible dialog trails of the system—yielding a test case coverage of 100%. Each poster represented a sub-part of the application domain and could be reviewed separately. The review was performed by one domain specialist in a three-day workshop, meaning on an average less than 3 minutes for each test case, where a test case averagely consists of 16 findings and 43 solutions (including the intermediate and the final solutions).

Figure 6 shows a picture taken during the second review of the knowledge base. The use of colored DDTrees, their printing on paper, and the use of the textmarker pen for the review process was perceived to be very intuitive by the domain specialist that performed the evaluation. Since no computer was required the (almost unexperienced) domain specialist could start immediately to work with textmarker and pen. The intuitive "user interface" was beneficial for erroneous areas of the tree. For example, when identifying errors the domain specialist could simply write/draw some text/corrections on the paper, e.g., linking a question to another sub-tree by drawing the edge manually on the poster, making comments etc.

5. Conclusions

We conclude the paper with a summary of the presented work by highlighting the most important aspects, and we discuss a number of issues for future work.

5.1. Summary

Empirical testing denotes one of the most important and frequently applied evaluation methods for knowledge systems. In this context, the interactivity of the system and the online derivation of solutions yielding early and intermediate solutions are distinct features of today's systems. Consequently, we motivated that the classic notions of a test case and the corresponding evaluation measures, such as precision and recall, are not sufficient to evaluate the interactive and online behavior of knowledge systems in an appropriate manner. We consequently enhanced the notion of a test case by the introduction of rated test cases and sequential test

cases. Both enhancements are true generalizations of the standard test case notion. The rated precision/recall and the chained and rated precision/recall were introduced as suitable extensions of the precision and recall measures in order to take into account the enhancements of rated test cases and sequential test cases, respectively. Also, the rated version as well as the rated and chained version of the precision/recall are true generalizations of the standard measures.

Evaluation measures often have only a small utility when not provided with appropriate inspection methods. We introduced a unit testing metaphor that uses the described measures in a 1-dimensional visualization, and additionally introduced the novel visualization technique DDTree, that depicts the results of a test run by a 2-dimensional rendering of a tree. The visualization of a DDTree can be used to effectively gain an overview of the overall "healthiness" of the knowledge base, but also is an intuitive instrument for the manual validation and debugging of new and faulty test cases.

The practical application and effectiveness of the presented measures and their use in the DDTree visualization was demonstrated in a case study. Here, the quality of the medical decision-support system Digitalys CareMate was evaluated in iterative sessions. The visualization of the knowledge and the test cases, respectively, was experienced as intuitive and effective.

5.2. Future Directions

Empirical testing builds on mature research and practical tools, but still is not a closed research area; there still remain a couple of issues to consider for future work. We briefly describe three issues in the following: 1) The extension to temporal case testing, 2) the integration in robustness tests, and 3) the adaptation of test case generation techniques.

Especially in the medical domain knowledge systems typically reason over temporal data, for example, weaning systems, e.g., the SmartCare system [30], or systems to monitor patients in the ICU, e.g., such as described in [32]. The testing of temporal knowledge systems was not explicitly discussed within this paper, but it can be seen, that the particular sequences of a sequential test case can be interpreted as particular sessions of a temporal consultation. This does not hold for temporal data that is asynchronously fed into the system, for example, in different time shifts and with varying frequencies. Here, it needs to be discussed if and how the notion of sequential test cases needs appropriate adaptation.

Robustness testing [5, 19] is an extension of empirical testing, that evaluates the valid behavior of the knowledge base with respect to either incorrect findings or a partly faulty knowledge base. Thus, the developed knowledge base is tested

for its reasonable behavior in noisy environments. The noisy environment is simulated by so-called torture tests, that gradually decrease the quality of the findings or the quality of the used knowledge. Then, the changed quality is tested by an empirical testing run. In the past, torture tests usually applied the standard measures for precision and recall, but it is easy to see that the degradation of the test cases with respect to their quality can be adapted to the new notions of rated test cases and sequential test cases. Furthermore, the analysis of the robustness test outcomes yields more interesting results when the extended measures for precision and recall are used.

In the past, alternative approaches for the automated generation of test cases have been presented, cf. [18, 20, 27, 28, 51]. These methods are useful for the generation of appropriate test cases in arbitrary domains, but they require the availability of explicit knowledge, either represented as already formalized knowledge (e.g. rules) or described as generation knowledge (e.g. constraints or causal dependency models). Furthermore, the possibility of rated solutions and sequential test data is typically not considered at all. In [6] the exhaustive generation of all possible (sequential) test cases was sketched using a simple *dialog bot*. Since this naive algorithm will generate an exponential number of possible cases, reasonable constraints need to be defined in order to shrink the number of combinations sufficiently.

Acknowledgements: The author thanks Dr. Wolfgang Lotz for contributing constructive comments and suggestions during the evaluation phases of the Digitalys CareMate project, and Marc-Oliver Ochlast for helping with the implementation of the final empirical testing tool. The presented work was partially funded by Digitalys GmbH.

References

[1] Adeli, H., 1990. Knowledge Engineering Vol I: Fundamentals. McGraw-Hill, Inc., New York, NY, USA.

[2] Akers, S., 1978. Binary decision diagrams. IEEE Transactions on Computers 27 (6), 509–516.

[3] Ayel, M., Laurent, J.-P., 1991. Validation, Verification and Test of Knowledge-Based Systems. Wiley.

[4] Barr, V., 1999. Applications of rule-base coverage measures to expert system evaluation. Knowledge-Based Systems 12, 27–35.

[5] Baumeister, J., Bregenzer, J., Puppe, F., 2006. Gray box robustness testing of rule systems. In: KI'06: Proceedings of the 29th Annual German Conference on Artificial Intelligence, LNAI 4314. Springer, pp. 346–360.

[6] Baumeister, J., Menge, M., Puppe, F., 2008. Visualization techniques for the evaluation of knowledge systems. In: FLAIRS'08: Proceedings of the 21th International Florida Artificial Intelligence Research Society Conference. AAAI Press, pp. 329–334.

[7] Baumeister, J., Seipel, D., 2006. Verification and refactoring of ontologies with rules. In: EKAW'06: Proceedings of the 15th International Conference on Knowledge Engineering and Knowledge Management. Springer, Berlin, pp. 82–95.

[8] Baumeister, J., Seipel, D., 2010. Anomalies in ontologies with rules. Web Semantics: Science, Services and Agents on the World Wide Web 8 (1), 55–68.

[9] Baumeister, J., Seipel, D., Puppe, F., 2004. Using automated tests and restructuring methods for an agile development of diagnostic knowledge systems. In: FLAIRS'04: Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference. pp. 319–324.

[10] Baumeister, J., Seipel, D., Puppe, F., 2009. Agile development of rule systems. In: Giurca, Gasevic, Taveter (Eds.), Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches. IGI Publishing.

[11] Beck, K., 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley.

[12] Bergmann, R., 2002. Experience Management: Foundations, Development Methodology, and Internet-Based Applications. Vol. LNAI 2432. Springer.

[13] Boroday, S., Petrenko, A., Ulrich, A., 2008. Test suite consistency verification. In: EWDTS 2008: Proc. of the 6th IEEE East-West Design & Test Symposium. pp. 235–238.

[14] Coenen, F., Bench-Capon, T., 1993. Maintenance of Knowledge-Based Systems. Academic Press.

[15] Djelouah, R., Duval, B., Loiseau, S., 2002. Validation and reparation of knowledge bases. In: ISMIS '02: Proceedings of the 13th International Symposium on Foundations of Intelligent Systems. Springer-Verlag, London, UK, pp. 312–320.

[16] Gamma, E., Beck, K., 1998. Test Infected: Programmers Love Writing Tests. Java Report 3 (7).

[17] Gómez-Pérez, A., 1999. Evaluation of taxonomic knowledge on ontologies and knowledge-based systems. In: KAW'99: Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Management.

[18] Gonzalez, A. J., Dankel, D. D., 1993. The Engineering of Knowledge–Based Systems – Theory and Practice. Prentice Hall.

[19] Groot, P., ten Teije, A., van Harmelen, F., 2003. A quantitative analysis of the robustness of knowledge-based systems through degradation studies. Knowledge and Information Systems 7 (2), 224–245.

[20] Gupta, U. G., Biegel, J., 1990. A rule–based intelligent test case generator. In: Proceedings of the AAAI–90 Workshop on Knowledge–Based System Verification, Validation and Testing. AAAI Press.

[21] Hartung, R. L., Håkansson, A., 2007. Automated testing for knowledge based systems. In: KES 2007: Proc. of 11th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part I. Springer, pp. 270–278.

[22] Helfman, R., Dumer, J., Hanratty, T., 1995. TED-turbine engine diagnostics: an expert diagnostic system for the M1 Abrams AGT1500 turbine engine. In: 11th Conference on Artificial Intelligence for Applications. IEEE Computer Society, Los Alamitos, CA, USA, pp. 1032–1038.

[23] Horiyama, T., Ibaraki, T., 2002. Ordered binary decision diagrams as knowledge-bases. Artificial Intelligence 136 (2), 189–213.

[24] Hüttig, M., Buscher, G., Menzel, T., Scheppach, W., Puppe, F., Buscher, H.-P., 2004. A diagnostic expert system for structured reports, quality assessment, and training of residents in sonography. Medizinische Klinik 3, 117–22.

[25] Kimura, M., Sakamoto, M., Adachi, T., Sagara, H., 2005. Diagnosis of febrile illnesses in returned travelers using the PC software GIDEON. Travel Medicine and Infectious Disease 3 (3), 157–160.

[26] Knauf, R., 2000. Validating Rule-Based Systems: A Complete Methodology. Shaker, Aachen, Germany.

[27] Knauf, R., Gonzalez, A. J., Abel, T., 2002. A framework for validation of rule-based systems. IEEE Transactions of Systems, Man and Cybernetics - Part B: Cybernetics 32 (3), 281–295.

[28] Knauf, R., Spreeuwenberg, S., Gerrits, R., Jendreck, M., 2004. A step out of the ivory tower: Experiences with adapting a test case generation idea to business rules. In: FLAIRS'04: Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference. AAAI Press, pp. 343–348.

[29] Ligeza, A., 2006. Logical Foundations for Rule-Based Systems. Studies in Computational Intelligence 11. Springer.

[30] Mersmann, S., Dojat, M., 2004. SmartCare$^{\text{tm}}$ - automated clinical guidelines in critical care. In: ECAI'04/PAIS'04: Proceedings of the 16th European Conference on Artificial Intelligence, including Prestigious Applications of Intelligent Systems. IOS Press, Valencia, Spain, pp. 745–749.

[31] Miah, S. J., Kerr, D., Gammack, J., Cowan, T., 2008. A generic design environment for the rural industry knowledge acquisition. Knowledge-Based Systems 21 (8), 892–899.

[32] Miksch, S., Horn, W., Popow, C., Paky, F., 1994. Intensive care monitoring and therapy planning for newborns. In: AAAI 1994 Spring Symposium: Artificial Intelligence in Medicine: Interpreting Clinical Data. AAAI Press.

[33] Milne, R., Nicol, C., 2000. TIGER: Continuous diagnosis of gas turbines. In: ECAI'00: Proceedings of the 14th European Conference on Artificial Intelligence. Berlin, Germany.

[34] Nalepa, G. J., 2007. Proposal of business process and rules modeling with the XTT method. In: SYNASC '07: Proceedings of the Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. IEEE Computer Society, Washington, DC, USA, pp. 500–506.

[35] Nalepa, G. J., Ligeza, A., 2005. A graphical tabular model for rule-based logic programming and verification. Systems Science 31 (2), 89–95.

[36] Nghia, D. D., Puppe, F., 2009. Hybrides, skalierbares Diagnosesystem für freie Kfz-Werkstätten [Hybrid and scalable diagnosis system for car garages]. KI: German AI magazine 23 (2), 31–37.

[37] Nilsson, N. J., 1982. Principles of Artificial Intelligence. Springer.

[38] Padma, T., Balasubramanie, P., 2009. Knowledge based decision support system to assist work-related risk analysis in musculoskeletal disorder. Knowledge-Based Systems 22 (1), 72–78.

[39] Preece, A., 1998. Building the right system right. In: KAW'98: Proceedings of Eleventh Workshop on Knowledge Acquisition, Modeling and Management.

[40] Preece, A., 2001. Evaluating verification and validation methods in knowledge engineering. In: Micro-Level Knowledge Management. Morgan-Kaufman, pp. 123–145.

[41] Preece, A., Shinghal, R., 1994. Foundation and application of knowledge base verification. International Journal of Intelligent Systems 9, 683–702.

[42] Preece, A. D., 1994. Validation of knowledge-based systems: The state-of-the-art in North America. The Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistomology 11.

[43] Puppe, F., 2000. Knowledge formalization patterns. In: PKAW 2000: Proceedings of the Pacific Rim Knowledge Acquisition Workshop. Sydney, Australia.

[44] Quinlan, J. R., March 1986. Induction of decision trees. Machine Learning 1 (1), 81–106.

[45] Richter, M. M., 2007. Foundations of similarity and utility. In: FLAIRS'07: Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference. AAAI Press, pp. 30–36.

[46] Santos, E., Dinh, H. T., 2008. On automatic knowledge validation for bayesian knowledge bases. Data & Knowledge Engineering 64 (1), 218 – 241.

[47] Seipel, D., Hopfner, M., Baumeister, J., 2005. Declarative querying and visualizing knowledge bases in XML. In: INAP/WLP'04: Applications of Declarative Programming and Knowledge Management (selected papers), LNAI 3392. Springer, Berlin, pp. 16–31.

[48] Smith, S., Kandel, A., 1994. Verification and Validation of Rule-Based Expert Systems. CRC Press, Inc., Boca Raton, FL, USA.

[49] Thompson, C. A., Mooney, R. J., 1994. Inductive learning for abductive diagnosis. In: AAAI-94: Proceedings of the 13th Annual National Conference on Artificial Intelligence. Vol. 1. pp. 664–669.

[50] Vermesan, A., Coenen, F., 1999. Validation and Verification of Knowledge Based Systems. Theory, Tools and Practice. Kluwer Academic Publisher.

[51] Vignollet, L., Lelouche, R., 1993. Test case generation using KBS strategy. In: IJCAI'93: Proceedings of the International Joint Conference for Artificial Intelligence. Morgan Kaufmann, pp. 483–489.

[52] Zacharias, V., 2008. Development and verification of rule based systems – a survey of developers. In: RuleML '08: Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web. Springer-Verlag, Berlin, Heidelberg, pp. 6–16.

[53] Zacharias, V., Abecker, A., 2007. On modern debugging for rule-based systems. In: SEKE'2007: Proceedings of the 19th International Conference on Software Engineering & Knowledge Engineering. pp. 349–353.