Learning statistics with episodic examples

A case-based reasoning system for R

Bachelorarbeit

im Studiengang Angewandte Informatik
der Fakultät Wirtschaftsinformatik
und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg
(2014)

Verfasser: Alexander Werner
Gutachterin: Prof. Dr. Ute Schmid
Acknowledgements

I would like to thank my advisor Prof. Dr. Ute Schmid for arousing my interest to this topic as well as for the support on the way of creating this thesis. I also would like to thank the participants of the user test, who have willingly shared their time during the processes of testing and interviewing. Finally I would like to express my gratitude to my family, who have supported me throughout my entire academic studies in so many ways.
Abstract

*Keywords: Case-based reasoning, episodic examples, analogy, problem solving, intelligent tutoring systems*

The technological advance demands for understanding and using technology even non-technical academic fields. For example statistical programming usually is a core element in the social sciences. While solving programming tasks novices tend to use similar, prior examples. Cased-based reasoning systems offer a good basis for accessing and using these episodic examples. In this thesis a Hi-Fi prototype for a case-based reasoning recommender for the statistical programming language R was designed and implemented. Subsequently the prototype was evaluated with regards to usability issues, the utility of the application and the usefulness of the predefined and episodic R examples. As a result of this the utility of the application and both example types has been determined though latter were helpful in different ways. In conclusion, the idea of such case-based reasoning systems appears to be promising and should be pursued further on.
# Contents

1 Introduction  1

2 Learning by example and case-based reasoning  4  
  2.1 Problem solving and case-based reasoning  4  
  2.2 Learning by example  5  
  2.3 Components of case-based reasoning  6  
  2.4 Intelligent tutoring systems  9

3 Conception of a CBR system  12  
  3.1 Use cases  13  
  3.2 Lo-Fi prototype  14  
  3.3 Case base  15  
    3.3.1 Case contents  15  
    3.3.2 Case structure  17  
  3.4 Similarity  18  
    3.4.1 Structural and semantic features  18  
    3.4.2 Information retrieval  19  
    3.4.3 Similarity measures  20  
    3.4.4 Overall similarity  21  
  3.5 Program structure  22

4 Realisation and Evaluation  23  
  4.1 Realisation  23  
    4.1.1 Case base  23  
    4.1.2 Connecting R to Java  24  
    4.1.3 Hi-Fi prototype  25  
    4.1.4 Similarity measures  26
4.2 Evaluation ................................................................. 29
4.2.1 Test setting ......................................................... 29
4.2.2 Results ............................................................... 31

5 Conclusion and future work ......................................... 33

References ........................................................................ 38

Appendix .......................................................................... 41
Chapter 1

Introduction

“Nam non solum scire aliquid, artis est, sed, quædam ars etiam docendi.”

Not only is there an art in knowing a thing, but also a certain art in teaching it.

(Cicero, De Legibus, II. 19.)

This quote implies how challenging it was to choose the right method and materials for education in Ancient Rome. More than 2000 years later nothing has changed. Though technology has evolved and tons of scientific findings were made since then, it is still a major subject of modern research. Moreover, technology and education are converging more and more and the technological capabilities are utilised for educational purposes. The usage of laptops, digital whiteboards and tablets in schools almost belongs to everyday life nowadays.

Due to these developments two conclusion could be drawn. First of all there exist demands for understanding and using technology. This is why in Germany projects in MINT\(^2\) fields have been facilitated in the last years. On a base level this is accomplished automatically by growing up with certain technologies. The knowledge of using a computer mouse was not natural thirty years ago, but it is today. Nevertheless more advanced knowledge like programming are not as widespread as needed. In most academic fields it is necessary to have a deeper understanding of certain software tools and computer aided methods. One of these academic fields are the social sciences. While most students expect a rather unmathematical subject, one part of this subject consists of statistical computing. Because the target group for this project are exactly those kind of students the statistical program-

\(^1\)http://en.wikiquote.org/wiki/Knowledge (retrieved Sep 2, 2014)

\(^2\)mathematics, informatics, natural sciences and technology
ming language R (see The R Foundation for Statistical Computing, 1991-2011) was chosen as learning content. R is an open source software (Ligges, 2008) which is used more and more in the field of social sciences.

As a result, this means technology should be the matter of education. The second conclusion which could be drawn from the evolving technological developments is that it has to be carried on utilising technology for education purposes. Many technological features are used nowadays to foster teaching and learning. Online courses enable easy sharing of learning contents, multimedia support offer different kinds of access to these contents and learning could be done asynchronous to others and more individually. One of those asynchronous technologies are intelligent tutoring systems (ITS) which provide individual contents to a learner. ITS are usually based on concepts from the field of artificial intelligence. One of these approaches is called cases-based reasoning (CBR). CBR means to solve new problems on the basis of former problems and their solution. This approach overlaps with analogical problem solving which is rather allocated to the cognitive sciences.

**Goal and structure of the thesis**

This thesis aims to suggest an approach on how CBR could be used to build an application which might assist students from the social sciences while solving programming tasks in R. This assistance is realised through the presentation of their own prior examples (episodic examples). Though R already offers code examples, these examples are not personalised and quite difficult to understand for novice programmers. This is why a comparison of their own examples to these predefined examples should be considered in this thesis. As a consequence the usage of findings from cognitive sciences to conceptualise an application is another demand of this thesis.

Due to that fact the subject of problem solving which is allocated in the social sciences will be described as the first part of this thesis. It depicts how problem solving is connected to analogies and how analogical problem solving overlaps with case-based reasoning. In addition to this it is described how examples and especially episodic examples may foster the learning process. Apart from that a closer look is taken at the components of CBR and at intelligent tutoring systems for programming.
Subsequently the conception for a CBR system is presented. After the definition and illustration of use cases their influence on the development of a low fidelity prototype is described. This description is followed by the consideration of a suitable case base which includes thoughts about its structure and contents. As a last part of the conception process an approach for measuring similarity is presented with regards to cognitive principles.

Following this chapter the state of realisation of the CBR system is described which includes the implementation of the case base and the attempts to connect Java to R. On this basis a high fidelity prototype was created which is depicted subsequently along with the description of the implementation of several similarity measures. The final part of this chapter pictures the evaluation of the high fidelity prototype and the resulting outcomes.

The final part concludes the approaches and results of this thesis and presents further challenges for this application.
Chapter 2

Learning by example and case-based reasoning

2.1 Problem solving and case-based reasoning

Learning a programming language is a kind of problem solving and needs a lot of practice to increase the number of experiences. A common heuristic to expand the search space in order to solve problems is the usage of analogies (Dörner, 1987). This method can be differentiated into within domain analogies (Dejong, 1989) and between domain analogies. A popular example of an analogy between two domains is the Rutherford analogy. In this example the domain of the solar system is compared to the domain of a hydrogen atom. The terms of within domain analogy and case-based reasoning overlap to a bigger part. The distinction exists, because the same paradigm is explored in different academic disciplines. In cognitive sciences it is more usual to use the term within domain analogy in the computer sciences it is mostly the term case-based reasoning.

As described above the term of a case might consist of a problem and its solution although the degree of formalisation may vary (Richter & Weber, 2013). Kolodner (1992) says that CBR can mean the recall and adaption of old solutions in order to solve new problems. Beside that definition Weber (1994) differentiates according to the purpose of CBR between classificatory and problem solving CBR. While classificatory CBR is focused on the classification and interpretation of new cases, problem solving CBR considers the process of adaptation of old problems to generate an appropriate new solution.
2.2 Learning by example

A common method to teach complex domains is problem based learning. This is mostly realised by presenting a written problem description. A task then consists of the description of the problem and its solution. In order to solve new problems analogue or earlier examples can be used as instruction to the new problem. Domains in which such instructions are used are mathematics or physics. If the solution is given step by step the instruction is called worked examples and has proved to be very effective especially for novices in the domains as mentioned above (Sweller & Cooper, 1985). In order to make structural similarities visible it is advisable to present at least two earlier examples to solve a new problem (Gick & Holyoak, 1983).

Another problem solving domain which is similar to these domains is programming. Compared with the domains of mathematics and physics they share features like a very structured and formal solution. Moreover, the solutions are structured hierarchical because in most cases some subproblems have to be solved first and in the correct order. Besides, the tendency of usage of old examples is a well documented phenomenon in learning a programming language (Weber, 1994). This is why case-based reasoning might play an important role in learning a complex domain.

Episodic examples and learning R

Another feature which must be considered while learning from examples is their similarity to the current problem. There may appear two different types of similarity. On the one hand the similarity may rely on structural features on the other hand on surface features (Holyoak & Koh, 1987). Common theories in analogical problem solving state that surface features play their key role while retrieving a suitable example while structural features are important when comparing the examples (mapping) and transferring the solution (Gentner, 1983; Holyoak & Thagard, 1989).

In addition to these statements some research was done to investigate further influence of surface features (Ross, 1989). In this field the term reminding is often used to describe the recollection of a similar problem which was experienced before. The term refers to the work of Schank (1982). These remindings might be seen as
In previous research the influence of these episodic examples was tested experimentally in the domain of the statistical programming language R (Werner, 2014). In a first programming session the students were introduced to the basics of the programming language R. This was realised in a tutorial which consisted of theoretical input and twelve programming tasks. In the second session the students had to solve five more programming tasks and were advised to use two programming examples for each task. For each task the students were assigned to one of two different conditions. The experimental group had to use two episodic examples from the first session to solve the new problem. The other condition contained two examples which were isomorphic to the old examples but differed in features such as the text and the names of variables. The hypotheses of this experiment was that surface features of an example have not only influence on the process of retrieving an old case but also on the efficiency and the effectiveness of its usage. In terms of CBR we would talk about the retrieval and the reusage of old cases.

The results had shown that there was no significant difference between the two conditions. Furthermore the results even tended to confirm an opposite effect. An explanation of these results might be the fact that the novice’s solutions of the first session which had been used in the experiment condition were not as clearly as their isomorphic pendants.

Despite these results old cases are a big source for instructions and should not be ignored. As a consequence of these results it might be helpful to clean up old cases by removing faults or generalise similar examples and present the structure explicitly to the user. With respect to the knowledge level of the novice user it might be an advantage to begin with very explicit examples and descend to more abstract formulas later on.

### 2.3 Components of case-based reasoning

The process of case-based reasoning might be separated into smaller components. One very fine grained approach is described in Weber (1994). At first there is the a new problem as input and the assignment of indices. This is necessary because the new problem has to be tested for relevant features to be compared with old problems.
Based on similar or dissimilar features the retrieval can take place. Similar indices might lead to stereotypical cases while dissimilar indices might lead to very specific cases. Subsequently the most suitable cases have to be selected by comparing the similarity of the new problem and the old case. To be able to do this a measure for similarity has to be calculated.

Riesbeck and Schank (1989) describes this step implicitly by subsuming it into the retrieval process. However Weber (1994) demands to make this step explicit because in most cases it is critical due to the computing time. After selecting a relevant case a new solution has to be constructed. This step is called adaptation because in most cases it is necessary to adapt the solution instead of making an exact copy. The adaptation may apply to the structure of a case and its attributes or the planning sequence of subproblems.

If a solution is successfully evaluated in a loop of testing, explaining and repairing the case and its indices are stored in the case base. An overview of these events are illustrated in figure 2.1.

As depicted in figure 2.1, Kolodner (1993) describes the indexing problem as a two-part problem as well. The first part is concerned with assigning the indices to the cases. In this step the situation must be described effectively by the indices to make a successful recall possible. The second part of the problem appears at the time of the recall where the new case must be elaborated in such detail that the relevant indices in the case memory can be found.

A further, more common characterisation of CBR is given by Aamodt and Plaza (1994). They separate the process of CBR into the four components retrieve, reuse, revise and retain which may occur repeatedly in this order as illustrated in figure 2.2. An additional fundamental issue of CBR is the representation of the cases.

**Case representation** The case representation covers two main aspects of CBR. The first is how to organise the cases the second is which features of a case should be stored. These aspects are fundamental because the following processes like the retrieval of old cases and integration of new ones depend on how the cases are represented (Aamodt & Plaza, 1994). The attributes of a case can be a flat, one dimensional feature vector or organised hierarchically on several dimensions. They also might either consist of structured or unstructured information. Depending of the extent of the case base the choice of relevant case attributes might also have effects on an efficient retrieval.
Retrieval The retrieval process is concerned with three further tasks. First of all, the features of a new case have to be identified. Possible issues might be the amount and kind of features which are accessible and how to deal with unknown features. As a second step, the identified features have to be matched against the old cases. This process can be divided into two steps. The first step is a kind of superficial preselection to reduce the case base to a set of cases which fit the problem at all. The last step contains a deeper analysis of the cases and the selection of the case which might be the most helpful to solve the problem (Aamodt & Plaza, 1994).

Reuse After retrieving a suitable case for the problem, the case might be used in two ways. The most simple way to make use of the old case is to copy the solution and ignore the differences. Another approach is to adapt the old case to the new problem (Aamodt & Plaza, 1994).

Revise After reusing and adapting a solution, it has to be evaluated. If the solution is not or just partially correct, it is necessary to detect and analyse possible errors (Aamodt & Plaza, 1994).

Retain The purpose of the last subprocess of CBR is to retain the new solved problem. This means to put the new case into the case base. It is therefore possible to extract different features of the case. These features may contain relevant descriptors of the problem and its solution, the reasoning path, and possible failures. After that all or just the relevant features are used to build an index to structure the search space for retrieval. As a last step, the index structure of existing cases might be changed due to the experiences with the new case (Aamodt & Plaza, 1994).

CBR knowledge models

Richter and Weber (2013) allocated the knowledge used in a CBR system to four containers based on the assumption that a CBR system is a knowledge-based system. The first container is called vocabulary and retains explicit descriptions of how the knowledge elements are used. The second one is called similarity, because it contains information about the similarity of the cases like a set of rules for determining the level of similarity of two cases. Other information stored in
2.4 Intelligent tutoring systems

A common combination of CBR systems in the domain of learning is the implementation of intelligent tutoring systems. This means to enrich a learning environment with computer assistance. This assistance might be based on artificial intelligence features which are well illustrated by Park Woolf (2008). Table 2.1 recapitulates these features. How these features might fit to this approach is described in the next chapter.

Psotka, Massey, and Mutter (1988) proposed that the ITS are heading into the direction of good personalised teachers. As a requirement of this development they emphasize the role of analysing teaching systems in a naturalistic setting. This approach focuses on both knowledge of experts and novices. They also state that
an ITS cannot replace a human tutor but augment a learning situation through personalised instructions.

Examples of programming tutors

There already exist tutoring systems which are concerned with learning programming languages. Most of these languages are relevant for students of computer sciences. One language which is important in the field of artificial intelligence is LISP. Anderson and Reiser (1985) have developed a tutor system for this language. Another systems to learn the programming language LISP is ELM-PE (Weber, 1994). In this system the knowledge of an user is represented in rules and concepts. Furthermore cases are stored as episodic entities, which contain information about used rules and concepts and information about the context of their usage. Another construct are plans which contain the right order for applying the rules. Later on those episodic entities will be generalised to retrieve known concepts. To handle cases with new concepts the rules must also be generalised. If a concept already exists the episode is integrated according to its structural and semantic similarity. The comparison of plans and code fragments lead to a measure of structural similarity while concepts and rules represent semantic similarity.

The retrieval process in ELM-PE uses a similarity measure which prioritises the structure of an episode and the position in its hierarchical representation. Not until then are semantic features considered as part of the similarity measure. This priority of structural features are consistent with observations from cognitive sciences as mentioned above (Gentner, 1983; Holyoak & Thagard, 1989).
Figure 2.1. *Flowchart of case-based reasoning*. Figure from Riesbeck and Schank (1989).

Figure 2.2. *Components of case-based reasoning*. Figure from Aamodt and Plaza (1994)
Chapter 3

Conception of a CBR system

This approach claims not to be an ITS despite the fact that some of the features mentioned by Park Woolf (2008) fits the definition. For example the storage of episodic examples applies to the feature generativity, because it provides hints customized to the user. Mixed initiative can be provided by offering to call hints actively as well as listening to the users input. Interactive learning can be realised by a relevant setting for the user which means to select meaningful problems as programming tasks. Besides it would lack some essential features like a model of the student or inferences over the given solutions.

Therefore the goal of this work is to start building a CBR system to support learning the programming language R. The primary task is to focus on the user which should interact with the system in order to retrieve suitable cases for new programming tasks. The adaptation of an old solution to a new case should be fulfilled by the user while a human tutor might correct the solution. There are several aspects to be considered in the conception of a CBR system. A first step is to select the right kind of CBR system which fits the aspired goal. There are several different types such as planning or knowledge intensive systems which rely on ontological knowledge or recommender systems which retrieve suitable solution from the case base and present them to the user. In the case of analogical problem solving a system which retrieves and provides analogues to a user would be sufficient. This means that the focus is on retrieving and retaining cases. The reuse of the cases is left to the user. To limit the system a little further let us assume a tutorial setting in which the revision of the solutions is also left to the user or a tutor. Further considerations are the kind of case representation, a similarity
measure and the behaviour of the CBR system especially its interaction with the users. The next sections will demonstrate these aspects.

3.1 Use cases

To outline the interaction of a user and a system it is useful to formulate explicit use cases which define the possible interactions. As a first draft several use cases were formulated and are illustrated in figure 3.1. Beside the user, who is called learner in this context, two further actors must be defined to illustrate the system’s behaviour properly. These actors are the tutor, who might bring additional knowledge into the case base as proposed by Psotka et al. (1988), and the R engine, which evaluates the learners input and is required for the R example call.

There are three use cases concerned with getting a hint from the system. The first is call R example which includes enter function name. This means that when a certain function name is set the corresponding examples from the R documentation are gathered and displayed. If the learner wants his own example he uses call own Example to get similar problems and their solution. The use case get last cases was designed for a special setting in which the program is used in a tutorial with bottom-up tasks. In this case the last case might always be the most helpful one. To test inputs the use case make R input is needed which includes evaluate input with the R engine as actor. After successfully solving a problem the correct solution must be saved. This is contained in the use case enter solution. The use cases verify solution and tag problem might have the learner and the tutor as actor depending on the context. While verify solution is needed to ensure the correctness of a solution the use case tag problem provides the possibility to add additional knowledge like used concepts. Finally the use cases load cases and store cases are interacting with a case base and are triggered by the learner’s interaction.

While the use case diagram gives an overview of all use cases and their dependencies another method is useful to get more fine grained insights. The use cases also were documented in high ceremony form. The main advantage of a high ceremony template is the ability to check main and secondary scenarios to get a first detailed overview of the system behaviour. Figure 3.2 illustrates such a template.
3.2 Lo-Fi prototype

Many use cases are considered with user interaction. Based on these use cases a low fidelity prototype was designed. The software Pencil (Evolus Co. Ltd, 2008-2013) was used to create the prototype for visualisation and a first walk through. The results of the walk through and the corresponding changes are documented in the next chapter in the section about the high fidelity prototype. Figure 3.3 shows the draft of the Lo-Fi prototype and a description of its elements. A interactive version of the prototype with working buttons can be found on the attached CD in the attachment.

Furthermore some cognitive concepts were realised in the prototype. On the one hand there are two hint areas which are an implementation of the results that two cases are more effective while constructing effective analogies (Gick & Holyoak, 1983). Another result of cognitive principles is the differentiation of the use case call own example. While the button Get Hints calls two full examples the
UseCase #: 1  
Name: Call R example  
Context: Learner needs hint  
Primary Actor: Learner  
Precondition: R engine running, set function name,  
Trigger: Button - call R example  
Main success scenario: 1. query R example, 2. Function name is transmitted to R engine,  
3. R engine responds, 4. R output is displayed  
Secondary scenarios: 2.1 invalid function name, 2.2 display error  
Success post condition: R output is displayed  
Minimal post condition: CBR session is still running  
NFR:-

Figure 3.2. Use Case documented in high ceremony style

button MyExample just recalls the solution of an earlier case. While Get Hints provides maximum information for an analogy, MyExample is more comparable to the existing R examples from the documentation. Figure 3.4 depicts a R example call for the function sample.

3.3 Case base

3.3.1 Case contents

As a next step the contents of the case base had to be defined. One idea was to use the programming task of a former study (Werner, 2014) as cases. This choice comes with quite a lot benefits. First of all the programming tasks are already integrated in a tutorial for the target group of students of social sciences. This complies with the interactive learning feature defined by Park Woolf (2008). Moreover each programming task corresponds to a test case. An example for this correspondence is illustrated in figures 3.5 and 3.6. The tasks are formulated in German. Figure 3.6 represents a new programming task, figure 3.5 a task from the tutorial which is a useful hint for the new problem. These tasks mainly consist of three parts. The first part is a textual problem description followed by an area for the own solution. The last part is a goal description which describes what the output of the solution should look like.
1. Concepts: This area displays the correspondent concepts of a problem. Use case: *tag problem*.

2. Problem area: In this area the tasks are presented.

3. Solution area: At the end of a task the user should put the code solution in this area. Use case: *enter solution*.

4. Console: Here is were the user might test his code. Use case: *make R input*.

5. Hint area: These two text fields display possible hints for the problem.

6. Function input: To get hints for a specific functions the name of this function must be specified here. Use Case: *enter function name*.

7. Button *MyExamples*: This button should bring the two most similar solutions to the hint area. Use Case: *call own example*.

8. Button *call examples()*: This button should bring the R examples of a specific function to the hint area. Use Case: *call R example*.

9. Button *Get Hints*: This button should bring the two most similar problems and their solutions and to the hint area. Use Case: *call own example*.

10. Button *Get Last Cases*: This button should bring the two last problems and their solutions and to the hint area. Use Case: *get last cases*.

11. Button *Finish Task*: This button finishes a task and calls the next test case.

Figure 3.3. *Draft of the Lo-Fi prototype*
> example(sample)

```r
sample> x <- 1:12

sample> # a random permutation
sample> sample(x)
[1] 1 3 4 9 5 2 11 7 12 6 8 10

sample> # bootstrap resampling -- only if length(x) > 1 !
sample> sample(x, replace = TRUE)
[1] 10 6 10 8 3 6 1 12 6 10 7 4

sample> # 100 Bernoulli trials
sample> sample(c(0,1), 100, replace = TRUE)
[1] 0 1 1 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0
[38] 1 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0
[75] 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 0 0 1 0 1

[...]
```

Figure 3.4. Excerpt of the output for calling the `example` function with `sample` as argument.

### 3.3.2 Case structure

As seen above the cases from the case base are in a very rudimentary format and must be structured so that a CBR system can work with it. Consequently a set of attributes has to be defined which describe a case. First of all a unique identifier is needed to address a case. This means an attribute `CaseId` is needed. The problem and its solution form the core components of a case and should be stored separately in the attributes `CaseDescription` and `CaseSolution`. Moreover several other attributes might be useful to enrich the cases with additional information. As mentioned above a case can be assigned to certain concepts. This means that some programming or statistical paradigms are contained in each problem and its solution. The cases could be tagged with these concepts either by the learner or the tutor. For an example the programming task illustrated in figure 3.5 belongs to the tutorial concept `functions`. As a consequence an attribute `Concept` should be included. Contextual information is also relevant. One kind of contextual information is, for example, when a case was solved and, in the event of multiple learners, from whom. This is why attributes like the `Date` and `Username` should be attached. At last an attribute for additional information should be considered. This information
Übung 4 - Funktionen und Funktionsoptionen

# Aufgabe:
# Vor der Erhebung Ihrer Daten möchten Sie R dazu nutzen die Versuchspersonen Ihrer Studie
einer von 3 Gruppen zufällig zuzuordnen. Die Gruppennamen sollen lediglich aus den Zahlen
1, 2 und 3 bestehen. Insgesamt erwarten Sie 30 Teilnehmer. Erstellen Sie eine Variable
"gruppe", die für jeden Teilnehmer eine zufällige Gruppenzuordnung enthält.

# Bereits definierte Variablen: --

# Eingabe:

gruppe <- sample(c(1,2,3),30,replace=TRUE)

-------

# Ausgabe (Hinweis: Da es sich um Zufallswerte handelt, werden Sie eine andere
Kombination aus Werten in der Variable "gruppe" haben.)
#
# > ...
# > ...
# [1] 1 1 1 2 2 1 2 1 3 3 2 1 3 1 2 2 2 3 1 3 3 1 2 3 1 3 1

Figure 3.5. R programming task as part of a R tutorial to build a case base.

might contain abstract data which may occur while processing other attributes or
additional notes or keywords. As a result an attribute Keywords should be included
in the case structure as well. In summary this means that every case is represented
as a flat, one dimensional feature vector.

3.4 Similarity

As a next step it has to be considered which kind of information the case attributes
contain. Moreover the cases must be made comparable and this is why a measure
for similarity has to be defined.

3.4.1 Structural and semantic features

The information stored in a case can be differentiated into semantic and structural
features. Following the differentiation of Weber (1994), semantic information con-
sists of concepts and rules while structural information contains code fragments
and plans. If this differentiation is transferred to the used case features most of them seem to be of semantical nature. For example the attribute *Concepts* contains semantic information just as the *CaseDescription*. In contrast *CaseSolution*, with its code elements, is a structural feature. Besides this differentiation the attributes *Date* and *User* should rather be defined as contextual information, because they are not directly connected to the content of the problem and its solution. Before these features are compared they must be processed into an enumerable format.

### 3.4.2 Information retrieval

The field of information retrieval and natural language processing is concerned with textual information. They offer methods to extract relevant information from unstructured data. As a first step there is usually the process of tokenising the raw input. The unstructured data which occurs in the cases are the features *CaseDescription* and *CaseSolution*. In both cases the data could be divided into smaller pieces. While *CaseDescription* could be tokenised to a set of words *CaseSolution* might be reduced to a set of regular expressions. The usual next step is to
remove less meaningful elements from these sets. This means for the CaseDescription to remove stop words like articles or prepositions. Usually this is provided by stop word lists. The consistent continuation of these information retrieval methods would be to use stemming methods to reduce further semantic redundancy. In the case of the CaseSolution less meaningful elements might be the name of variables and arguments. These elements could be removed while checking the code tokens against a list of function names. At this point it has to be mentioned that tokenising the CaseSolution has the effect of turning it from a structural to a semantic feature. Besides this the other case attributes consist of structured data and can be transformed to sets very easily. These sets of tokens could be reduced to similarity measures by applying different similarity coefficients.

3.4.3 Similarity measures

Natural language processing is a domain which offers many methods to measure similarity between textual data. The principle behind this similarity measure is to tokenise the text to form two sets of words which could be compared by several similarity coefficients (Kapetanios, Tatar, & Sacarea, 2013). Three of these coefficients should be implemented. These coefficients could easily be applied to all case attributes if they have been transformed to sets. The following formulae 3.1, 3.2 and 3.3 illustrate these coefficients while X and Y are sets of words or other elements.

Matching coefficient:

$$sim_{matching} = |X \cap Y|$$ \hspace{1cm} (3.1)

The matching coefficient is defined as the cardinality of the intersection of the two sets.

Overlap coefficient:

$$sim_{overlap} = \frac{|X \cap Y|}{\min(|X|, |Y|)}$$ \hspace{1cm} (3.2)

The overlap coefficient is also defined as the cardinality of the intersection of the two sets. Additionally it divided by the size of the smaller set.
The Jaccard coefficient is very common in the field of text mining. While the dividend is still the cardinality of the intersection of both sets the divisor is the cardinality of the union of them. As a consequence the result is a value between 0 and 1.

### 3.4.4 Overall similarity

Weber (1994) has demonstrated that it is possible to use a measure of similarity based on cognitive principles. This means to prioritise structural features over semantic features. While the importance of each feature in the states of CBR is an ongoing debate a pragmatic approach in this sense could be considering both of them and think about different priorities in a later stage of development. Although there are no structural features left, caused by tokenising the CaseSolution, a generic formula was formulated as to how semantic and structural features could be calculated into one similarity measure. This formula is illustrated in (3.4). The principle behind this formula is very simple. First of all several similarity values of both types, semantic and structural, are calculated. Then their arithmetic mean is calculated separately and weighted with values between 0 and 1. Finally they are divided by 2 to get a similarity value between 0 and 1 if the jaccard coefficient had been used.

\[
\text{sim}_{\text{overall}} = \frac{\beta_1 \sum_{i=1}^{n} \text{sim}_{x}\text{(structural)}_i + \beta_2 \sum_{j=1}^{m} \text{sim}_{x}\text{(semantic)}_j}{2}
\]

\(\beta_1\) weight structural
\(\beta_2\) weight semantic
\(\text{sim}_x\) a similarity coefficient of type x
structural set of structural elements
semantic set of semantic elements
\(n\) number of structural similarity measures
\(m\) number of semantic similarity measures
3.5 Program structure

The structure of the program might be divided into several packages by their purpose referring to Richter and Weber (2013). A package named representation could contain the Case class which defines the case structure and corresponds to the case base container. In addition a similarity package should contain the measures for similarity. There is no plan to implements a vocabulary container because the knowledge elements are implicitly processed in the similarity measure. Moreover, steps of adaptations are not saved separately which makes a container for them obsolete from the start. Besides this structure a package for user and system interaction should be created to store GUI elements and the classes which are connected to the system.
Chapter 4

Realisation and Evaluation

4.1 Realisation

The implementation of the CBR system was developed in Java using Eclipse platform as integrated development environment (The Eclipse Foundation, 2000-2012). The phase of implementation was separated into several subtasks. As a start a representation for the case base had to be found. Another subtask was to get Java connected to R. After completing these two tasks the creation and a first evaluation of a high fidelity prototype with users was possible. As a last step the similarity coefficients mentioned in chapter 3 were implemented.

4.1.1 Case base

The use cases concerned with the case base are load cases and store cases. A first consideration while conceptualising a case base is the extent and the dimensionality of its content. Because of the one dimensional nature of these cases and a manageable extent of cases not a relational database was chosen but a simple text format to store the case base. As a result of this the case base consists of a csv file. To load and store the cases the additional library opencsv was used (Smith, Sullivan, & Conway, 2005-2011). After reading each case from the file an Object Case is created and put into a Collection while caseId serves as key. For a start every attribute is just a String and the class just contains getter and setter methods. A class diagram of the Case class is illustrated in figure 4.1.
4.1.2 Connecting R to Java

In order to realise the use cases call R example and make R input the application has to be able to evaluate R expressions. Moreover an R console should help the learner to interact with such expressions. A possibility to get a connection between Java an R is a framework called JRI (JRI (Version 0.5-4), 2007-2014) which is part of the rJava project (rJava (Version 0.9-6), 2003-2012). While rJava allows to evaluate Java expressions in R, JRI does the same the other way around.

Interactive console A first attempt was to create an interactive console within a Java GUI to make an execution of R possible through the software. The interactive textconsole works fine in the IDE and the output could be successfully transferred into a GUI element. But so far it was not possible to redirect Strings from a GUI element to the console as input.

JRI additionally offers the possibility to evaluate Strings which contain R code to regular expressions. For this approach the JRI R engine needs the data types of the expressions contained in the R code as additional information. This is applicable for single calls but for an interactive console it is quite complex because the Java object which contains the R code has to be parsed first to determine its data type.

To compensate the lack of an interactive console users have to use an R interpreter in an external terminal window to check their inputs. Their inputs are not logged for further processing. After solving a problem the solution has to be transferred manually to the GUI element in Java in order to save the solution.
Realisation and Evaluation

**Call R example** Apart from this issue a method to call R examples was easy to implement. The function name is set as argument to the R function `example()` and evaluated by an instance of an *R engine*. The method illustrated in listing 4.1 can be found in the package `userInterface` in the `Console` class.

```
Listing 4.1
call R example

```java
public String rCallExample(String input) {
    String evalString = null;

    // Create a stream to hold the output
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    PrintStream ps = new PrintStream(baos);

    // Backup System.out
    PrintStream old = System.out;
    System.setOut(ps);

    // Evaluate input
    reng.eval("example(" + input + ")");

    // Reset System.out
    System.out.flush();
    System.setOut(old);

    // Write output to String
    evalString = baos.toString();

    return evalString;
}
```

4.1.3 Hi-Fi prototype

After these two subtasks the creation of a Hi-Fi prototype was possible. Figure 4.3 illustrates the graphical user interface of the Hi-Fi prototype. In comparison to the Lo-Fi prototype (illustrated in figure 3.3) some changes were made as the result of a quick evaluation. The field for concepts is now more in focus in order to get more tags created by the learner. Moreover the arrangement of the buttons had been changed because the button for an R call belongs to the query field for function names. Due to the reasons described above the console element was removed. Instead of that an extra terminal window has to be attached to the main GUI element.
The use case log in was also realized through another GUI element which collects a user name in order to create an individual case base for the learner. This element is illustrated in figure 4.2.

Elements that were already implemented in the prototype were loading and storing cases and the call for R examples. For an evaluation of the Hi-Fi prototype the similarity of the cases was assigned manually. This was done because the evaluation of the prototype should be focused on user interactions and the experience of working with helpful hints and the predefined R examples.

In contrast to the Lo-Fi prototype the buttons MyExample, Get Hints and Get Last Cases are disabled in the beginning. After clicking the button Finish Task five times and finishing four tasks the buttons Get Hints and MyExample are activated for the last two tasks. While Get Last Cases is not implemented yet Get Hints shows the two manually assigned old cases consisting of the task description and the users solution. MyExample displays just the users solution.

### 4.1.4 Similarity measures

The calculating of the similarity between cases consists of several steps. As a first step the unstructured attributes CaseDescription and CaseSolution have to be tokenised into sets. The package java.util provides a class StringTokenizer which is suitable for tokenising the CaseDescription into a set.

In addition a list of German stop words was read and transformed into a set. An Iterator of this stop word set was used to remove all stop words from the set of CaseDescription tokens.

Transferring the CaseSolution to a set was implemented a little differently. The method illustrated in listing 4.2 gets the CaseSolution as a String object and iterates over a list of function names. The result is a set of function names which were included in the CaseSolution.

---

Figure 4.3. Screenshot of the Hi-Fi prototype
A list of all function names of all used R packages could easily be generated in R. Listing 4.3 shows what these commands look like. The first command returns a list with all function names of the `base` package. The second R command returns, amongst other things, a list of all loaded R packages. These two commands were used to write a Java method `rCallFunctionList()` located in the class `Console` which returns a list with all function names of all loaded R packages.

As a last step the similarity coefficients mentioned in the last chapter had to be implemented. All coefficients need the intersection of two sets which can easily be done by predefined methods of the `Collection` framework. While the method `retainAll()` returns the intersection of two sets, the method `addAll()` returns their union. The cardinality of a set is nothing more than its size. As a result, listing 4.4 illustrates the implementation of the Jaccard coefficient. With the implemented coefficients enables the calculation of the overall similarity like illustrated in listing 4.5. The implementation of this and the other coefficients can be found in the `similarity` package in the class `Similarity`.
4.2 Evaluation

4.2.1 Test setting

The Hi-Fi prototype of the CBR system was evaluated by two former students of psychology with no experience in programming and R. This was done to have the software tested by its target group to get representative results. Before these students were able to solve programming tasks they needed to be taught some contents of R. This was realised through a part of the tutorial which was used in a former study (Werner, 2014). The used part of this German tutorial can be found on the CD in the appendix. As a part of this tutorial they had to solve four programming tasks using the prototype. In this phase a case base was built. After a ten minute break the two test persons had to solve two tasks more. At this point they were allowed to use all kind of hints.

While solving those final tasks the users were asked to use the think-aloud method. This means speaking out loud all kind of thoughts which may occur during the task. To record their thoughts and behaviour a screen recording with audio has been created. This was realised with the open source software tool kazam.

### Listing 4.4

Jaccard coefficient

```java
public float jaccardCoefficient(Set<String> str1Tokens,
                                Set<String> str2Tokens) {

    Set<String> unionSet = new HashSet<String>();
    Set<String> intersectionSet = new HashSet<String>();

    unionSet.addAll(str1Tokens);
    unionSet.addAll(str2Tokens);

    intersectionSet.addAll(str1Tokens);
    intersectionSet.retainAll(str2Tokens);

    float union = unionSet.size();
    float intersection = intersectionSet.size();
    float result = intersection / union;

    return result;
}
```
Listing 4.5
Overall similarity

```java
public float overallCoefficient(Case case1, Case case2, int simtype,
    float beta1, float beta2) throws IOException {

    float result = 0;
    int amountOfSemFeatures = 2;

    // semantic features
    Set<String> case1DescToken = removeStopwords(stringToTokenSet(case1
getDescription()));
    Set<String> case2DescToken = removeStopwords(stringToTokenSet(case2
getDescription()));
    Set<String> case1SolToken = extractFunctionNames(case1.getSolution());
    Set<String> case2SolToken = extractFunctionNames(case2.getSolution());

    // jaccard coefficient
    if (simtype == 0) {
        result = (beta2 * (jaccardCoefficient(case1DescToken, case2DescToken) +
            jaccardCoefficient(case1SolToken, case2SolToken)) / amountOfSemFeatures)
            / 2;
    }
    // overlap coefficient
    if (simtype == 1) {
        result = (beta2 * (overlapCoefficient(case1DescToken, case2DescToken) +
            overlapCoefficient(case1SolToken, case2SolToken)) / amountOfSemFeatures)
            / 2;
    }
    // matching coefficient
    if (simtype == 2) {
        result = (beta2 * (matchingCoefficient(case1DescToken, case2DescToken) +
            matchingCoefficient(case1SolToken, case2SolToken)) / amountOfSemFeatures)
            / 2;
    }
    return result;
```
In addition the application *keymon* was used to make mouse clicks visible (Kirkwood, Lin, Taylor, & Steiner, 2009-2014). Subsequent to the user test a short semi structured interview was conducted. The questions were concerned with positive and negative feedback to the prototype, the requirements of the users, the practicability in a tutorial setting and the utility of the different hints.

### 4.2.2 Results

The user tests revealed that the *R example* was not the first choice for a hint because the function name has to be recalled actively. Even after a short break of ten minutes the users had difficulties to recall the exact function names. Moreover they had problems to recognise the relevant information from an R example if it was called successfully. Nonetheless one user reported that even if this kind of hint is a little inconvenient it is good as a kind of encyclopaedia for functions.

The usage of *Get Hints* was experienced ambivalently. On the one hand these kind of hints had a big recognition value, which makes it easier to reconstruct an old solution. However both users experienced this hint as too extensive. One user was afraid of getting too much information and of undermining a learning effect. The other user recognised the old task but not his own solution.

As a consequence the hints for *MyExample* had been used for several reasons. One of them is the clearness of the hints. The user is not overwhelmed with information. A second reason is that, contrary to the R examples one function, but several functions in a composition are displayed.

The data of the corresponding interview question confirms these results. While *R example* has its purpose for precise requests, *MyExample* offers the best balance between recognition value and clearness. One user experienced these different hints as a possibility to solve the problem step wise for a maximum learning experience.

Additional observations were multiple clicks on the same hint button. This could be traced back to the explorative setting of the test. Another explanation, which was confirmed by data from the interview, is that the user does not know which of the hints is active. Furthermore the users had not been informed that they would see their own solution again. As a consequence one user would have had written more textual information into the solution. In addition to these findings some usability issues had been detected. First of all the R console was not very
usable at all. An empty input led to a prompt to quit and after every typing error the whole expression had to be written again because of an absent command history. This lack was even more severe due to different operating copying methods. While copying from the console needed to be done by mouse the solution area of the software operated only by shortcut.

The results of the interview revealed even more usability weaknesses. One user asked for a scalable font size and colours which are more appealing. Moreover feedback after each completed task was suggested in order to motivate the learner. Altogether the adjustment of the GUI elements had been experienced as well-arranged.

One of the last questions asked the users for their opinion concerning advantages and disadvantages of such a system in a tutorial setting. The concordant answer was that the program allows the user to practice at his own speed. This might be a clear advantage if a learner misses to hook up with the tutorial. More positive effect are that the exercises are documented automatically. In order to use the program for autodidacts, though, some additions such as more feedback, error correction and smaller exercises will have to be made.
Chapter 5

Conclusion and future work

To conclude this thesis the former work and results will be summarised and interpreted. A positive outcome is that users from the target group see the advantages of such an application in the context of learning the programming language R. To learn at an individualised speed and with examples which had been processed before, are a great benefit. Besides these findings the predefined R examples still have their right to exist, because they are needed as a kind of encyclopaedia. Another outcome was that even episodic examples should be given stepwise to foster the learning situation and to provide clear hints. This result complies with findings in the field of worked out examples (Sweller & Cooper, 1985). An idea to implement this is to represent the CaseDescription in several parts for task, goal and further notes. Another idea which came up as a result of the user test was the creation of a glossary of used functions which are illustrated by episodic examples.

In addition to these findings much work is to be done to elaborate this application. Several usability issues arose during the user test which have to be solved. The implementation of a usable interactive console would be a great step forward, because user inputs could be analysed during an interaction with R. This would enable the program to start modelling the users knowledge which would be a great step further to a real ITS. Furthermore the usage of structural data should be considered by parsing the solutions less rigorously.

Another task to be completed is to evaluate the similarity measures with respect to the case base. An interesting question is whether the measure provides the same results as a manual assignment of the cases. As a consequence of these results it should be considered to connect the application to existing CBR frameworks.
like jColibri (Recio-García, 2008) in which implementations of similarity measures already exist. In conclusion, it can be noted that finding the right way to teach a thing still remains an art which can not be mastered by an application yet.
List of Figures

2.1 Flowchart of case-based reasoning ........................................ 11
2.2 Components of case-based reasoning ..................................... 11

3.1 Use case diagram .............................................................. 14
3.2 Use Case documented in high ceremony style .......................... 15
3.3 Draft of the Lo-Fi prototype ................................................ 16
3.4 Excerpt of the output for calling the example function with sample as argument ......................................................... 17
3.5 R programming task as part of a R tutorial to build a case base. . 18
3.6 R programming task as a testcase ......................................... 19

4.1 Class diagram for the Case class .......................................... 24
4.2 Login GUI element ............................................................ 26
4.3 Screenshot of the Hi-Fi prototype ........................................ 27
List of Tables

2.1 Artificial Intelligence Features of Intelligent Tutors . . . . . . . . . 9
Listings

4.1 call R example .................................................. 25
4.2 extract function names ........................................ 28
4.3 list of all function names of the package ”base” and list all loaded packages ........................................ 28
4.4 Jaccard coefficient .............................................. 29
4.5 Overall similarity .............................................. 30
References


Appendix

CD

This thesis comes along with a CD which contains the following:

- A digital version of this document
- Use case document
- An interactive Lo-Fi prototype
- An interactive tour through the GUI elements of the Hi-Fi prototype
- Implementations in R
- Implementations in Java
  - Hi-fi prototype (executable)
  - Program source with documentation in Javadoc
- Evaluation
  - Materials for the R tutorial
  - Interview questions
  - Recorded Interviews
  - Screen cast of the user test with think aloud protocol
Erklärung:
Ich erkläre hiermit gemäß § 17 Abs. 2 APO, dass ich die vorstehende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Bamberg, den 11.09.2014

__________________________

Alexander Werner