Case based Reasoning Approach for Re-use Activities

M. Zinn[1,3], K. P. Fischer-Hellmann[2] and Ronald Schoop[3]

[1]University of Plymouth, Drake Circus, Devon, PL4 8AA, Plymouth, U.K.
[2]University of Applied Science Darmstadt, Haardtring 100, D-64295 Darmstadt, Germany
[3]Schneider Electric Automation GmbH, Steinheimer Str. 112, D-63500 Seligenstadt, Germany

Abstract. The development of software applications is partly or entirely based on the re-use of software units. For software engineers, this leads to the problem that it is not possible to know all processes, technologies and supporting applications and the alternatives needed for the re-use of a software unit. As a result software engineers are not able to employ the most optimal solution known. Based on case based reasoning this paper outlines a way to use the stored knowledge of a specific re-use activity in order to give software engineers assistance if they want to perform similar activities. This solution consists of a proposal system for a re-use activity information system. The publication concludes with the result that it is possible to re-use, within a given an environment, specific knowledge for other integration activities.

1 Introduction

The re-use of software units is one of the major topics of software engineering. At the same time this topic is also a wide area of scientific research. One of the central questions of this research field is to find a consistent description of software units. The answer to this question is e.g. associated with the following objectives [1]: (1) saving of resources (time and effort), (2) reduction of know-how and greater flexibility when re-using software units.

Due to this question, in the past decades, many different methods and technologies have been developed for the re-use of software units. As an example of current approaches that promote re-use, object orientation, component-orientation and service orientation are mentioned [2].

One of the problems of re-use is to define what a re-usable software unit is [3]. From the conventional view that only the part of a software unit that is actually used again (e.g. binary or source code) is such a re-usable software unit, the trend was created that also additional information (such as documentation, specification, test information, etc.) can be used again. Because of this diversity of information the terms 'assets' or 'artifacts' are used [4]. As a result a re-usable software unit thus includes many different needs for information within a re-use process.

This diversity poses a problem in answering the scientific question. The complexity of the problem increases because for each re-use technology additional methods

Zinn M., P. Fischer-Hellmann K. and Schoop R..
Case based Reasoning Approach for Re-use Activities.
DOI: 10.5220/0004177900310043
In *Proceedings of the 3rd International Workshop on Software Knowledge* (SKY-2012), pages 31-43
ISBN: 978-989-8565-32-7

and applications supporting the re-use technology were developed. This strong expansion of data or information is called information explosion [5]. Software engineers have to find their way in this confusing environment.

Potential solutions can be found in the area of knowledge management (KM). KM and information systems (IS) are able to organise knowledge and information to structure and deliver it consistently [6]. Technologies such as semantic models allow the connection of different elements, creating knowledge-based statements about the knowledge of this relation. A typical example of such knowledge relation is found today in social networks and advertising. Social networks are capable of grasping knowledge entered by the user and generating adverts that might interest the user, based on this knowledge. The selection of advertisements is based on the knowledge entered by previous users. For the social networks the operator of knowledge generation is created using an added value. This process is known as 'Knowledge Harvesting (KH)' [7].

In principle, the method of KH may also be used in the re-use of software units. This means that knowledge about an existing software unit or a related re-use activity can be used to generate statements for other software units or activities. An existing IS or KM system that is capable of generating software units and their knowledge of software re-use activities and save it to reproduce (to perform it automatically), will be extended. This extension allows for generating predictions about alternative methods and technologies or any other specific application systems that can be used in a software re-use process. The prediction execution is focused in this publication.

2 Problem Identification

Since the scientific question has been not answered and the objectives are not implemented there is the problem that software engineers may have a comprehensive knowledge of all existing re-use technologies and the associated methodology and supporting software applications. In the following the problem of missing knowledge on the methodology and supporting software applications in the re-use of software units is focused. Usually this knowledge is specialised with certain (re-use) technologies or development models. Software engineers typically obtain this knowledge through a learning process. The experience of a software engineer supports him/her in making decisions about the re-use of software units. However, a person acquires this knowledge only when he/she works with such methodologies or applications, or is informed or somebody shows them it. This process is referred to as learning process. If a software engineer wants to solve a sub problem of software re-use (partly) automated, he/she can only do this by knowing about the corresponding application that solves the problem. Applications that a person does not know about are in this case not part of the solution approach designed by the person or the amount of knowledge of the person. This problem can be shown based on the information demand model for the re-use of software units [8].

Fig. **1** shows the structure of the Information Demand Model for the re-use of software units. It demonstrates the problem that a person lacks knowledge about a specific step of the re-use of a software unit. The subjective information demand

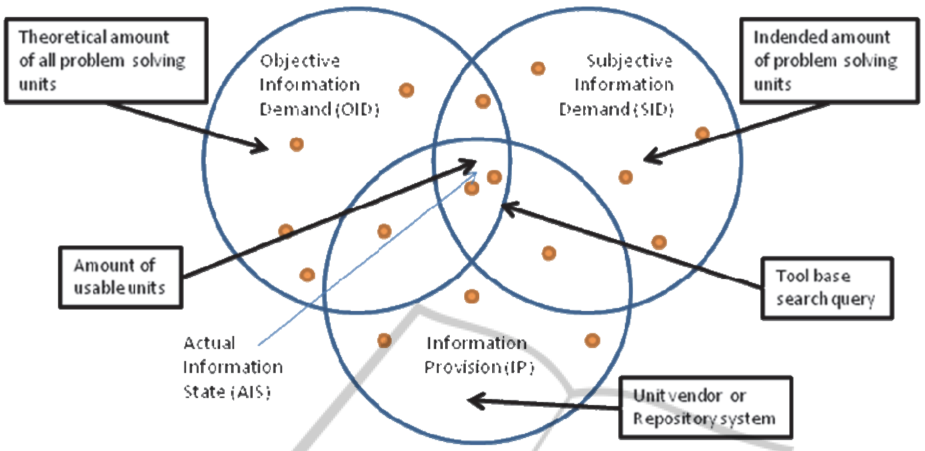
Theoretical amount of all problem solving units
Objective Information Demand (OID)
Subjective Information Demand (SID)
Indended amount of problem solving units
Amount of usable units
Tool base search query
Actual Information State (AIS)
Information Provision (IP)
Unit vendor or Repository system

Fig. 1. Information Demand Model [8].

(SID) shown in Fig. **1** contains all solutions a person can imagine. The overlap of all three areas is the solution sets, which recognises a person who is theoretically correct (OID) and are being offered or for the person actually reached (IP). This solution set will be formulated by the inability of an individual to search even more restricted (IQ) [9]. It can be assumed that the amount of actual usable solutions (intersection of all three areas) is much lower than the approaches located in the overlapping area between the objective need for information and the offered solution sets. The reasons for this can be explained as follows:

Younger software engineers do not have much experience and knowledge in software re-use. Interestingly, these people are very interested in the re-use of software units [10]. I.e. The problem of missing knowledge shown in Fig. **1** actually exists for them. Another reason for this problem is that software engineers often have tasks that deal with new technologies or new approaches. Knowledge on the new information is usually limited to those persons concerned [10].

It is assumed that the knowledge of various software engineers is different. For example, it may be that a person is an expert in a service-oriented technology. Another person, however, is rather an expert in the use of object-oriented technologies. Both persons are experts in their field and have worked in this field with the usual methods and applications that support them in the re-use of software units. When swapping these two individuals to each other's technologies it is expected that a certain learning process is necessary to achieve the same knowledge level of a real expert in that particular technology area.

The fundamental problem can be formulated as follows: Due to the fact that many technologies, i.e. methods, processes and software applications for the re-use of software units and related activities, exist (information explosion), the problem arises that a software engineer does not have the complete knowledge to (re)use all this technology. It raises the question how the necessary knowledge, to fill in for the missing part of the activities of re-using software units, can still be made available to a person.

3 Perspective Information and Knowledge

This publication deals with the fundamental question of how to transform data into information and information into knowledge. In addition this knowledge should be available to other persons. This refers to the range of knowledge elements of the re-use of software units. The scientific background of such an investigation can be illustrated using the data–information–knowledge–wisdom hierarchy (DIKW) [11]. Based on DIKW hierarchy the elements 'Data', 'Information', 'Knowledge' and 'Wisdom' are defined as follows:

"***Data*** *are defined as symbols that represent properties of objects, events and their environment. They are the products of observation. But are of no use until they are in a useable (i.e. relevant) form. The difference between data and information is functional, not structural.* ***Information*** *is contained in descriptions, answers to questions that begin with such words as who, what, when and how many. Information systems generate, store, retrieve and process data. Information is inferred from data. Knowledge is know-how, and is what makes possible the transformation of information in to instructions.* ***Knowledge*** *can be obtained either by transmission from another who has it, by instruction, or by extracting it from experience. […] Wisdom is the ability to increase effectiveness.* ***Wisdom*** *adds value, which requires the mental function that we call judgement. The ethical and aesthetic values that this implies are inherent to the actor and are unique and personal.*" [11]

Software units within this publication represent 'Data'. The DIKW hierarchy uses data to generate information, if a relation of the individual data elements (e.g. parts of the software units) is made to each other. The range of information on a software unit includes all possible information about this unit, such as the description of the technical contents, unit structure, technological information, and information about authors or producers etc. Information turns into knowledge if information is so far connected to each other that it can be used to perform an activity. As part of the re-use of software units, this means that information about a software unit for a user is brought into relation to the extent that these users transform the unit, for example, or integrate it into a development environment (can carry out its activity trap). This last step in the scope of software unit re-use represents a scientific problem [12] and is focused on in this paper.

The area of 'Wisdom' is the next step in the processing of knowledge. It is about clearance from the perspective of knowledge to do the right thing. But this step is not part of this publication and is not discussed further. But it is a long term problem in the re-use of software units and should be discussed and resolved.

4 Solution Approach Definition

In the following, an approach to solving the problem described above is outlined. It deals with re-use activities of integration, transformation and deployment of the re-use of software units. I.e. The outlined approach is able to store information about software units and bring it into relationships. This constitutes knowledge and can be used to describe the above-mentioned activities and perform them with technical support.

An information system that allows saving information and knowledge about software units and re-use activities ([13], [14], and [12]) is used as a base system in this study. Through modeling of re-use activities (e.g., transformation of a software unit or integration of a software unit in a development environment) within this system, it is possible to store such activities of a particular software unit within the information system [12]. By defining and using a service-oriented environment the information system capable is of automatically performing these activities. This means that a user who has knowledge of the software unit and specific activities may deposit this knowledge in the information system. A user, who does not have this knowledge, can use the information system to access this knowledge and use it, even without learning the knowledge. The usual scenario using this information system is explained in more detail in the following example:

Example: A user who is an expert in web technologies has stored a web service software unit in the information system that is able to sign files. As information about the web service, he/she defines, among other things, that the software unit consists of a web service description file (WSDL) and a text document that serves as documentation on the software unit. In addition, the system needs some meta information (e.g. author and functional description) for this software unit, which serves, among other things, to carry out a semantic search for this software unit. After the input of the actual software unit, the expert user defines a transformation activity. He/She indicates that a particular software application (svcutil.exe) from Microsoft is able to transform the software unit (WSDL) file into a source code file containing an implemented web service client. For this transformation, he/she must also specify which information is needed for the transformation. In this case, there are various parameters and the WSDL file of the software unit. The expert also needs to define, that the result of the transformation is a new software unit. After entering this transformation and configuration into the information system, the system is able to offer another user information about a software unit (e.g. download of the software unit and its documentation) and the execution of the related activity (in this case, the transformation of the WSDL file into web service client as source code. Another user searches for a web service that is able to sign files and gets the information from the system including the loaded software unit present. The user can now view the stored metadata and the software unit. In addition, he/she is able to download the data from the software unit. The user can also view information about the stored transformation. Here it is shown that the transformation yields results. The transformation can now be performed by 'pressing on a button' within the information system and the result is delivered to the user to download.

The proposed information system is able to store re-use knowledge about specific software units and perform it. Basically it can be said that the above described problem has been solved. Users without knowledge can perform activities (i.e., unknown applications and methods) with the required knowledge. This statement is only correct if it is assumed that a person wanted to use an application in order to achieve a particular result. However, this person has no knowledge of the application used to get into the activity or methodology. The person now knows that such an activity can be performed. While this is an important factor of the fundamental information for re-use, such a user is not able to define the same activity for a similar software unit in the information system or carry it out.

Considering Fig. **1**, it can be stated: A person is only able to enter re-use knowledge in the information system, if he/she knows (has learned) these activities (knowledge and knowledge application). Conversely, it can be said that a person who has no knowledge of this cannot lacks the scenario depicted a way that knowledge on other similar tasks or activities on be transferred.

4.1 Extension of the Existing Solution Approach

During an experiment [12], of which the goal was to underpin the approach of the information system shown in Example 1, the authors recognised that the knowledge of the activities entered by the participants can be used in another context. This knowledge can be used in a predictive system to support people who create new activities. This approach is hereafter called the 'Predictive Software Re-use Activities (PreSRA)' and follows a simple principle. The entered information about an activity, which is the input, the output and the characteristics, are stored as patterns within the information system. A user can choose three different ways to work with it. **Search for Activities:** The user can explicitly search for an activity within the information system for a software unit. For this purpose, it determines the type of activity as well as the familiar input and output information. The information system then analyses previously entered activities on this model and can give the user a recommendation, which is already a recorded activity to fit its defined input and output. **Automated Proposal System:** When creating an activity for a software unit, the user have to define an application or select an application known to the system that performs automated activity, e.g. transformation from the application example 1. In addition, the user must define the input and the output of an activity. With this definition, the information system can automatically detect the pattern for this activity based on the user input and compare them with existing patterns in the system. The result is a list of alternative applications which are able to process this pattern or alternative configurations for the already selected application. **Free Use:** Based on the second point 'Automatic proposal system' can be defined using another variant. Users can use the information system for activities by entering: (1) Input parameters for an activity and / or (2) the desired output and the desired result of the activity and/or (3) specific information or browse the properties of an activity. The result is an outlined list of possible activities that a user can use. Unlike the first two variants here it is not the goal to find an activity in the information system to re-use a software unit, but to preserve the knowledge of how an activity can be simulated. Such knowledge can be passed to the user i.e. in textual form.

This PreSRA approach also supports user input activity knowledge or users who generally want to identify an application that is capable of performing a certain activity at a certain given pattern. The problem described above will now be solved with the approach. It is noted that not only the automated execution of an activity without knowledge is possible, but also the knowledge that is necessary for the performance users will be provided as a suggestion system. Here are different ways to use this special knowledge. There are three interesting 'proposal variants': Proposal system for the integration of software units in development environments, proposed transformation system for applications, and proposed system for device-based deployment. This division into three systems proposal does not represent the full amount of any possible systems for the re-use of software units, but represents the focus of this publication. The basic study aimed at finding [15], integration [14], and transformation and a special case of the deployment of so-called embedded devices [16]. The topic of integration is focused in this publication.

4.2 Technical Structure of the Approach

The PreSRA approach will be explained using the example of integration of software units into development environments and the existing previously outlined information system. This system will be now explained. This is necessary to understand the context of the data used by the proposed systems.

4.2.1 Information System Architecture

The central core of a basic information system is a data model that uses semantic relationships. This data model is able to store information about software units. Different components of software (plugins) have access this data model to perform different tasks. E.g. repository plugins allow the loading of units from different software repositories, which have different data models. This creates a unified view on different data sets within the information system. Plugins are able to perform re-use activities e.g. transformation and integration. An extension of the basic semantic data model is necessary. Communication plugins allow sending knowledge or information to clients / plugins for further processing (e.g. implementation of activities). It is therefore possible to view, download or use information about software units on other computer systems. At the same time it is possible to perform activities using the information system and send the result to (remote) clients. The information system provides its functionality by using the communication plug-ins in the form of various communication technologies (e.g. SOAP or REST based web service). The basic scientific investigation, however, focused on web service technology.

4.2.2 Used Data Model

The basic data model consists of four areas [15]. The first section describes metadata about the software unit, such as authors, support information, creation date, etc. The second section deals with the representation of the software unit as a solution or problem. The basic focus of the investigation is not on this scientific problem, but this area was reserved for further research in the data model. The third section describes the technical part of the software units. Takes place simultaneously in this area, a semantic model that the search of a software unit using a noun-verb combination allows [15]. The fourth section describes a software unit from a technical perspective. I.e. the contents of a software unit are defined by its physical data. Figure 2 shows this part of the data model.

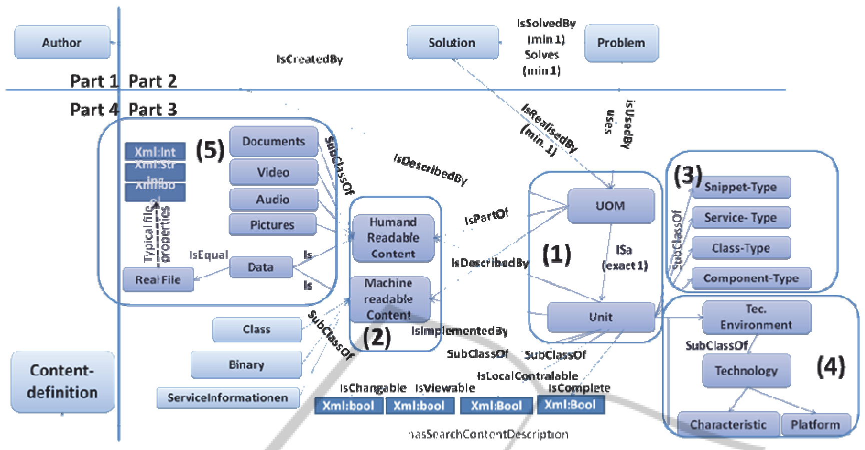

Fig. 2. Part 4 of the basic data model [15].

This area consists of five major sub-sectors. The area (1) represents the central element of the entire data model: a software unit. This item has relations with all other areas of the data model. The area (2) classifies a software unit for re-use technologies such as services, object, and components. The area (3) contains important features for the classification of a software unit from the perspective of the types described in section two. Thus, for example, a class is a software unit that is variable, accessible, complete and verifiable. Service, however, is a software unit is not changed, with privacy, however, is not completely controllable. The area (4) specifies the technical environment that requires a software unit. Usually at this point the technology platforms and environment can be defined. The field (5) defines a software unit as a physical file. The files are distinguished by their intended use. There are files that are readable for humans or are intended for systems.

All data model extensions (i.e. for integration activities) are linked to the part 3 of the data model, because of software units are the main element of each re-use activity.

Integration. This term describes the insertion of a software unit into a development environment. Today's development environments can include different options that can be inserted, such as a software unit. The data model's task hereby is to generalize this knowledge and present it in a way that can be processed if required by the information system. This approach has been demonstrated in a previous publication [14]. The data model extension shown in Figure 3 was designed to depict this type of integration.

Therefore, this process requires the integration of different files (File), a description of the development environment (IDE) and a description of the integration pattern. The integration pattern provides a uniform description of the development environments' different integration possibilities (see Figure 3, [14], and [12]).

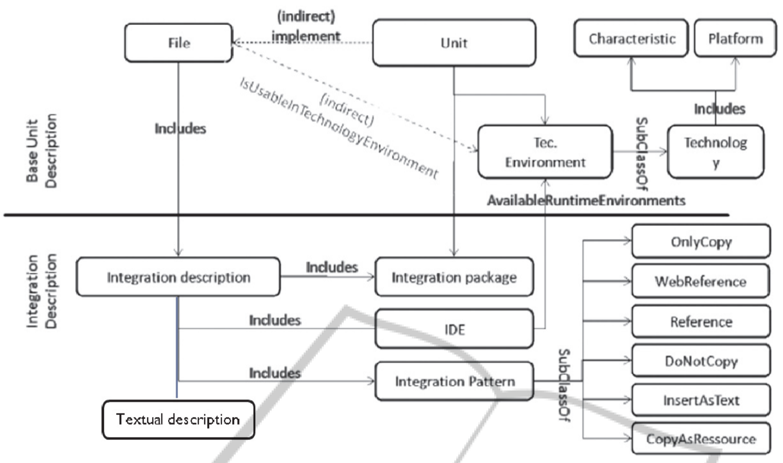

Fig. 3. Integration activity extension [14].

5 Information System as Integration Activity Proposal System

In the following the data model extension for integration of software units stated in the previous section is used to explain the concept of a proposal system. This paper introduces the basic data model and the expansion of the existing data model of information systems used for the storage of integrating knowledge. This includes the execution of integration on the basis of this knowledge [12]. For this purpose the data model shown in Figure 3 was used. The basic principle of the information system is to generalize knowledge for a particular activity and furthermore, generalize knowledge about the use of standardized interfaces specifically in software units. Accordingly, the data model shown in Figure 3 is a generalization of various models that are required for execution of integration knowledge. The most important information for the integration of a software unit is (1) Which files of a software unit can be used and (2) How these are integrated into a development environment. This not only directly affects the files of a software unit but also influences their possible dependencies. Figure 3 (1) describes the presented information and specifies the files to be integrated. These are files belonging to the software unit whose dependencies have to operate in the development environment. These dependencies can be part of the information contained in the Information system (for example, additionally stored files) or files or environment variables that must be part of the development or runtime environment of the system. In this area, (2) the information shown in Figure 2 comprises the information of the development environment. Among other things, this describes which type of development and runtime environment and which associated configurations are needed for the unit. Furthermore, this demonstrates a classification set that specifies the fundamentals of the way in which a software unit is integrated. The following content samples can be derived from this information (see Figure 3):

Table 1. Information of integration activities objects.

Typ	Describtion	Symbol
File(s)	All files participating in the integration process. This includes all kinds of information available about each file, eg Technology, type, name, size, etc.	File Set<File>
Integration type(s)	The integration of each file type participating in the integration process. This includes all information shown Figure 3.	Type Set<Type>
IDE(s)	Any development environment described by means of the data model. This includes any information, e.g. Name, supported technology platform, environmental variables, system files, operating system	IDE Set<IDE>
Dependency(ies)	Each dependency of a file, e.g. Technical environment, platform, environmental variables, system files, folders and file structures, relation to other software units, etc.	Dep Set<Dep>

Due to the pattern shown in Table 1, the following content model relationships can be derived from the data model shown in Figure 3[1]:

Table 2. Input and output patterns table for integration activities.

		Input pattern			
		Type, Set<Type>	File, Set<File>	IDE, Set<IDE>	Dep, Set<Dep>
Output pattern	Type, Set<Type>	(✓)	✓	✓	✓
	File, Set<File>	✓	(✓)	✓	✓
	IDE, Set<IDE>	✓	✓	(✓)	✓
	Dep, Set<Dep>	✓	✓	✓	(✓)

By entering one of the input patterns defined in Table 2 can one can specify a corresponding output pattern. This will be illustrated by the following example:

While entering the files a person wants to integrate, i.e. Information, Technology, type, name, size, etc., the information system is able to compare this with stored knowledge of previous integrations. This process discovers which of the stored files contain identical or similar information. An output pattern can be used from the resulting quantities of suitable integrations as this creates information integration type, used IDE and further necessary dependencies. This includes the information content of each sample, e.g. IDE platform. This process is called case-based reasoning [17].

This type of search can be used for each of the input pattern's elements defined in Table 2. In addition, elements of the input pattern can be logically linked to obtain a more accurate result. This is illustrated by the following example. The information system can be asked with which IDE it is possible to integrate necessary files and how to define the integration type of each file specifying how these should be integrated (see Table 3). All saved integrations will be compared to see whether a similar

[1]Due to the semantic relationships of the entire data model other input and output patterns can be identified. The results presented in this work are patterns and therefore represent examples that serve for direct use. This also applies to the patterns shown for other usages.

process to arrive at the content of these files can be provided to others with the same type of integration. The resulting set of integrations indicates that such integration is possible and specifies which IDEs can be performed.

Table 3. Search patterns.

		Input pattern			
		Type, Set<Type>	**File, Set<File>**	**IDE, Set<IDE>**	**Dep, Set<Dep>**
Input (& relation)	**Type, Set<Type>**	not useful	not useful	✓	maybe not useful
	File, Set<File>	not useful	not useful	✓	maybe not useful
	Result			✓ **(& relation)**	

6 Discussion of the Focused Approach

As part of the solution to the problem of 'making knowledge available to people', this publication presents two problem approaches. The first solution is the basic information system that was considered and discussed earlier from the perspectives [13] and [12]. This system provides access to information and the execution of activities through the use of stored knowledge, which is also available through remote communication systems. With this approach, the following objectives in relation to the basic problem are to be met: (1) Users need no knowledge of the software repository, including the repository's location as well as the means to access and operate it. (2) Users require no knowledge to perform an activity. The information system and the expansion of automation plug-ins enable the integration, transformation and deployment of software units. As discussed in the first section, knowledge can be used with this solution, even if a person is not aware of this. The idea presented in this publication extends the information system with a proposal system (PreSRA) which is able to work with the accumulated knowledge about integration. It is also to offer capability of such users to further knowledge based added value (i.e. transformation or deployment). There are three considered scenarios: (1) Automatic creation of activities: Due to the fact that the proposed system is able to compare input patterns with existing patterns, the system is also capable of generating a re-use activity from a given input pattern. Thus, for example by entry of file information, a comparison with other transformations can be performed. If transformations are found, the system can process these files and is able to create an automatic transformation from it. This transformation can then be verified by an end user. The system can then use these transformations in its knowledge base. (2) Search by activities: The examples showed in the previous section show that the PreSRA system can be used i.e. to search. This applies to any activity that a particular input pattern expenditures by testing the knowledge base to a greater or lesser amount of expenditure patterns. (3) Transfer of knowledge: Looking at the data model extensions for the activities of the integration, transformation and deployment are each composed of individual steps. Figure 3 shows a manual step description. A suggestion system may be adapted so that it not only stores automatically running activities in the information system, but can also serve as a step by step description. A user is then able to perform every single step of an activity manually. This helps the user learn the knowledge that is necessary for a particular activity. Besides the search for knowledge and execution of activities (i.e. integration), users are now able to define knowledge activities (i.e. integration) without having the appropriate knowledge. Additionally, they are also capable of using this knowledge to instruct other users. This solves the problem discussed above, that people without knowledge are unable to instruct others. This applies only within the scope of this paper and under the use of its proposed integration activity. In addition, this system allows the user to generate templates to enable other users to learn knowledge for re-use activities (i.e. integration).

7 Conclusions

This publication focuses on the problem of users inability to perform re-use integration activities of software units due to a lack of knowledge. Additionally, these individuals were not able to use existing knowledge to solve similar problems or to support other people. At the same time, this publication outlines a solution to these problems. An existing information system can (automatically) perform such re-use activities based on expert knowledge it received as input. This information system has been extended in this paper to analyse the input of knowledge and non-expert users can use it as a suggestion system. This enables users to ask the system for information, e.g. Software units in form of activities and/or the system can create such activities from existing activities or even execute them. In addition, it was shown that the system's knowledge of the activities can be made available to the user by using a case-based reasoning approach, which enables them to repeat these activities manually and thus acquire the knowledge themselves. This represents a solution for people with no knowledge, defined in the context of re-use activities like integration. This approach can be used for future research, including other activities, such as automated. Likewise, the problem may be the definition of Wisdom 'of knowledge from the perspective of re-focusing of software units'. It is also necessary to consider whether the method described before is applicable to other domains.

References

1. E. Henry and B. Faller, “Large-scale industrial reuse to reduce cost and cycle time“, IEEE Software, Bd. 12, Nr. 5, S. 47–53, Sep. 1995.
2. G. Wang and C. K. Fung, “Architecture paradigms and their influences and impacts on component-based software systems“, in 37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the, Big Island, Hawaii, pp.. 272–281, 2004.
3. I. Sommerville, Software engineering. Boston: Pearson, 2011.

4. R. Oliveto, G. Antoniol, A. Marcus, and J. Hayes, “Software Artefact Traceability: the Never-Ending Challenge“, pp. 485–488, 2007.
5. C. Alvarado, J. Teevan, M. S. Ackerman, and D. Karger, “Surviving the Information Explosion: How People Find Their Electronic Information“, Massachusetts institute of technology - artificial intelligence laboratory, Apr. 2003.
6. F. O. Bjørnson and T. Dingsøyr, “Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used“, Information and Software Technology, Vol. 50, No. 11, pp. 1055–1068, Oct. 2008.
7. N. Nakashole, M. Theobald, and G. Weikum, “Scalable knowledge harvesting with high precision and high recall“, pp. 227, 2011.
8. M. Zinn, K. P. Fischer-Hellmann, A. D. Phippen, and A. Schütte, “Information Demand Model for Software Unit Reuse“, presented at the ISCA 20th International Conference on Software Engineering and Data Engineering (SEDE-2011), Las Vegas, Nevada USA, S. 32–39, 2011.
9. A. Picot, Die grenzenlose Unternehmung : Information, Organisation und Management Lehrbuch zur Unternehmensführung im Informationszeitalter, Wiesbaden: Gabler, 2003.
10. S. G. Shiva and L. A. Shala, “Software Reuse: Research and Practice“, in Fourth International Conference on Information Technology (ITNG’07), Las Vegas, NV, USA, pp. 603–609, 2007.
11. J. Rowley, „The wisdom hierarchy: representations of the DIKW hierarchy“, Journal of Information Science, Bd. 33, Nr. 2, S. 163–180, Feb. 2007.
12. M. Zinn, K. P. Fischer-Hellmann, and R. Schoop, „Automated Reuse of Software Reuse Knowledge in an industrial environment – Case Study Results“, presented at the 17th IEEE Internation Conference on Emerging Technologies & Factory Automation, Krakow, 2012.
13. M. Zinn, K. P. Fischer-Hellmann, and A. D. Phippen, „Development of a CASE tool for the service based software construction“, presented at the 5th Collaborative Research Symposium on Security, E-learning, Internet, and Networking (SEIN’2009), Plymouth, 2009, pp. 134–144.
14. M. Zinn, K. P. Fischer-Hellmann, and R. Schoop "Reusable Software Units Integration Knowledge in a Distributed Development Environment“, Second International Workshop on Software Knowledge (SKY2011), pp. 24–35, 2011.
15. M. Zinn, K. P. Fischer-Hellmann, and A. Schütte, “Finding Reusable Units of Modelling - an Ontology Approach“, in Proceedings of the 8th International Network Conference (INC’2010), Heidelberg, 2010, pp. 377–386.
16. M. Zinn, K. P. Fischer-Hellmann, and R. Schopp, “Reuseable Software Unit Knowledge for Device Deployment“, in Conception of complex automation systems, Magdeburg, Germany, 2012.
17. B. P. Allen, “Case-based reasoning: business applications“, Communications of the ACM, Bd. 37, Nr. 3, pp. 40–42, 1994.