Automated Reuse of Software Reuse Activities in an Industrial Environment – Case Study Results

Marcus Zinn
University of Plymouth
Plymouth, UK
marcus.zinn@plymouth.ac.uk

Klaus-Peter Fischer-Hellmann
University of Applied Science
Darmstadt, Darmstadt, Germany
k.p.fischer-hellmann@digamma.de

Ronald Schoop
Schneider Electric Automation
Seligenstadt, Germany
ronald.schoop@schneider-electric.com

***Abstract* - The reuse of prefabricated software units, such as classes, components and services is one of the central topics of software engineering and requires lot of knowledge and experience. Instead of focusing on the knowledge management processes and a resulting lifelong learning process of individuals, this paper shows an experimental study based on an approach of automation of knowledge based reuse activities. This is done by employing a unified view of software construction activities and software units used by these activities in an industrial environment. It concludes that software engineers of different industrial business units and knowledge levels can be supported by performing different software construction activities with only one approach, the result of which avoids a long learning process for software engineers.**

Keywords-Automated software unit reuse; software reuse activities; industrial environment; case study.

I. INTRODUCTION

The reuse of software units (like classes, components, or services) requires professional knowledge or expertise. A software unit is a technical unit, and can, therefore, be defined like a software component in the context of this paper:

"*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition of third parties*". [1]

Typically, software engineers have to acquire this knowledge. In industrial environments, the knowledge depends not only on the technical properties of a software unit but also on the technical environment, technical topic (e.g., embedded devices) and the business topics (e.g., Automation, Datacenters, Mines & Minerals). Today knowledge about software units in a reuse context is a broad field. As adequate description of knowledge in the context of this paper following definition is used:

"*... the capability of a man (or an intelligent machine) to use information for problem-solving*" [2]

Starting from this point of view a software engineer has to have different kinds of information to perform software reuse, as for example: (1) Information about technical properties such as programming language, necessary technical environment, and dependencies. A software engineer has to know this information. [3]

(2) Information about interfaces and business context. A software unit solves at least one problem. Typically, the interfaces and provided data types are related to this fact. By handling such a software unit a software engineer have to be aware about this information. [3] (3) Information about the reusable artefact. Today a reusable software unit is more than a single binary file. Related information like test cases, documentation, and versioning are also reusable and sometimes implied. A software engineer has to deal with this related information. [4] (4) Information about related reuse concepts and processes. Software unit reuse is not undertaken if a software engineer decides to perform reuse. Many activities such as search, validation, integration, transformation, and testing are part of a reuse process. A software engineer must be aware of the existence of different reuse processes and technologies.

As a result of these perspectives, reusing a software unit may define as the use of different information about a software unit and a given environment to perform a number of reuse activities. The result is a reused software unit in a software development project.

Based on the high number of different technologies, business context, reuse artefact information and possible reuse concepts or technologies, the amount of necessary knowledge is high. This results in a problem for software engineers. Each time they wish to reuse a software unit they have to know about the relevant activities, and the related knowledge and information. If this knowledge is missing the reuse cannot be carried out successfully.

A solution may be the automation of reuse activities. As shown in the automation industry, this requires the development of supporting systems that are able to perform activities for a user. By automating software reuse activities, software engineers are able to perform these activities without having acquired the complete knowledge. Such an approach would reduce the problem of missing knowledge and was discussed in the past [5] and [6] under the name of "Service based Software Construction Process (SSCP)". However, the experimental proof of this concept is still missing.

This paper describes the setup and the results of the first

 ISBN: 978-1-61208-230-1

phase of an experiment validating the concept of SSCP, which is described by the following hypothesis:

"*Automated Software reuse activities will reduce the problem of missing knowledge in software unit reuse*"

This work forms part of the research on a Service-based Software Construction Process (SSCP) incorporating the field of Software Unit Reuse. The goal of this research is to identify a semantic model (about finding, adapting, integrating, and deploying of software units) combined with service technology that supports software engineers by performing software reuse (finding, adapting, integrating, and deploying) without having all needed information. The paper contributes to the research area by demonstrating the positive effect of automated software reuse activities, based on software reuse knowledge on the problem of missing knowledge in software unit reuse, in a real world experiment.

After the problem statement in the next section, the Section 3 shows the focused solution of this paper. This is used in Section 4 to describe the experiment setup and execution. Section 5 discusses the experiment results followed by the conclusion section (Section 6)

II. THE PROBLEM OF REUSE IN MULTIPLE INDUSTRIAL SOFTWARE DEVELOPMENT TEAMS

Typical aims of software reuse are to reduce costs and time in development projects [6]. These are two reasons why reuse of software units is an important part of software development in industrial areas [5]. However, the use of reuse in industrial projects does not guarantee a successful project, a fact, which has been demonstrated by several project studies in the past [6]. Typical problems are [6], e.g. ,: Misconceptions (reuse == repository, reuse == OO), No non-reuse specific processes modified, No reuse specific processes installed, No training/awareness actions, Reusable assets produced but then not used, Multi contractor / Multi company project, and No production of assets.

The last problem 'No production of assets' differs from the others. This problem deals with the fact that a software unit must be developed in order to be reusable [7]. If this is not the case, the amount of required resources is decreased by reuse [6][7]. Based on this statement, the effort to reuse increases after the creation of a software unit and should remain at the same value continuously for each reuse.

An internal study conducted by Schneider Electric [8] indicates a complex but interesting picture. A set of around 50 software units (so-called 'bricks' in industry area) has been created and widely reused. The average reuse number is between 9 and 10. The distribution of reuse for different bricks is shown in Figure 1. It starts with a minimum of 3 reuses (the point where typically a cost breakeven would start compared to a non reuse approach) and spans up to 36 reuses.

Relating to the above mentioned fact 'No production of assets' the study of Schneider Electric shows a dilemma of reuse in industrial environments. A reusable software unit creates additional reuse effort during the creation phase and in reuse phases of each development team which reuses this unit.

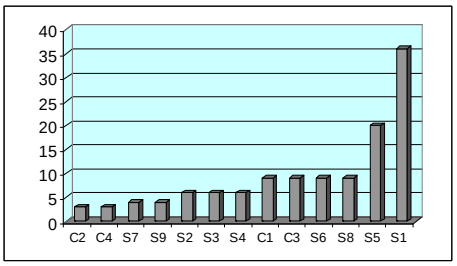

Figure 1. Distribution of reusable bricks [8]

Creation Phase Dilemma (CPD): The creation of reusable software includes different phases, which focus the reusability. Typical examples are given by Software Product Line approaches [7]: (1) Generalisation – The interfaces and functions of a software unit must be generalised to increase the reuse probability. (2) Integration – The software unit must be built in a way that it can be integrate in the development projects of other teams. (3) Support – The software unit must be 'equipped' with additional reuse artefacts, which support the reuse, e.g., reuses documentation. Additionally, such a unit have to be installed in a system, which provides access to it.
All of these steps require knowledge from an expert user.

Reuse Phase Dilemma (RPD): Each development team has now different challenges for reusing such a software unit. Typically, each team has to find and download the software unit [8]. In the next steps, they have to understand and integrate the unit into their development projects [7]. Sometimes software units must be adapted (transformed) for that specific application [9]. Figure 2 shows also the typical support and maintenance effort, which is created during these steps. This effort is the results from the problem that the development teams have not enough knowledge to perform the described reuse steps.

CPD and RPD are typical theoretical examples discus-

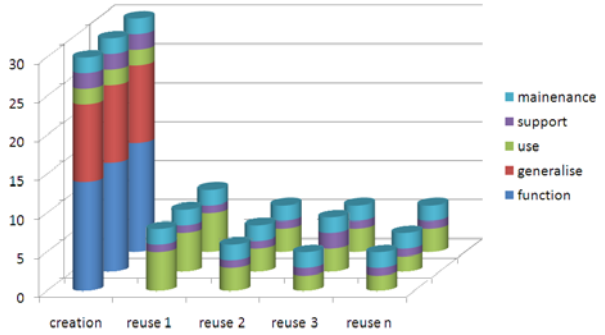

Figure 2. Support and maintenance effort [8]

 ISBN: 978-1-61208-230-1

sions of problems. The reality creates two additional dilemmas in the context of CPD and RPD. (1) **Creator dilemma (CD)**: The creation team is not available for support at the time of reuse (people are loaded with other projects or change team or organization) (2) **Reuser dilemma (RD)**: The reuse teams are different for each development projects, and therefore the exchange of a 'learning curve' between the teams is not possible.

Figure 2 shows that each development team has nearly the same problems and need nearly the same amount of resources. The challenge of reuse based software development in industrial areas is to reduce the sketched dilemmas. The purpose of this study is to show that reuse of a single software unit in multiple teams does not need this amount of resource on both sites: creator and reuser.

III. CONTEMPORARY SOLUTIONS

Nowadays, there are different approaches for the above-mentioned problems. The first approach is so called information systems, which, in general, enable the storage of information. This enables a user to search for information. However, such systems are not designed specifically to address the issue of transformation, but treat the subject of information generally [10]. Generally, such systems can be used to save information about an area of knowledge in textual form, but without the context of knowledge (see [10]). Each software construction activity may be described in this form and may be stored in an information system. The user is now faced with the problem of obtaining this information and interpreting it correctly in order to perform a successful transformation. Usually, information systems are not intended to apply their stored information automatically. But they can be extended for this task [10].

Despite this lack of functionality, information systems comprise a part of this article's advocated solution. Extensions of information systems are so-called Knowledge Base System (KBS) [10]. Such systems are defined as:

"*… a method that simplifies the process of sharing, distributing, creating, capturing, and understanding a company's knowledge*." [11]

Knowledge systems are not fundamentally designed for the subject of software construction activities. Furthermore, the authors of this article believe knowledge systems are missing a fundamental property: the automated application of stored knowledge for specific tasks. However, there is a lack of systems that have asserted themselves and are not focused on the typical software construction activities of software units. The latter property 'application of knowledge', is also a part of the solution discussed in this article. Basically, the knowledge that is necessary for perform an reuse activity can be stored in knowledge systems.

The area of software development has currently seen a number of interesting approaches dealing with specific subjects of a software reuse activity. Most of them are specific for one reuse activity type. For example there are two existing approaches for the activity of software unit transformation which are of interest: Model transformation [12] and generative programming [13]. Both approaches have existed for some time and form the basis for approaches that are being used today. Both support software engineers in generating reusable transformation models or rules. However, additional knowledge is necessary to make use of both approaches. This can be found in other activity areas like deployment [14] and Integration [15]. For the integration of software units into Integrated Development Environments (IDE) very specialised solutions exits e.g., Packaging for Eclipse or Packaging for Visual Studio. But these products are too specialised and require different kinds of specialised knowledge from the user.

The above mentioned solutions have one common problem. They assume a high learning curve. But learning how to implement every existing technology or solution for knowledge based problems cost too much time. It is necessary to identify a solution, which is able to support software engineers by performing software reuse activities without a lifelong learning process.

IV. FOCUSED SCENARIO

The basic idea of the targeted solution is that an expert applies knowledge (knowledge extraction) about the software reuse activity of a specific software unit to a system, which is able to perform the activity automatically with a minimum of human interaction based on knowledge. Users who do not have the necessary knowledge are now able to perform this activity (knowledge injection). A learning process for this specific activity and the specific software unit is not necessary. Figure 3 shows this scenario.

The idea was presented in previous [5][6] where its advantage was demonstrated for two reuse activity examples: Integration of software units into integrated development environments (IDEs) [15], and deployment of software units into embedded devices [14].

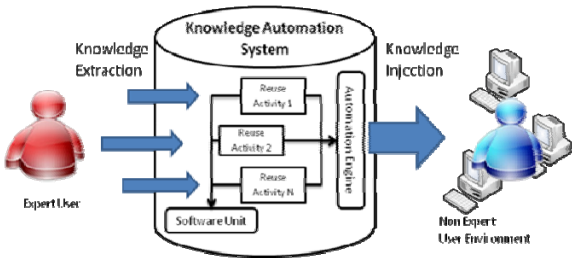

Figure 3. Concept of the focused solution

 ISBN: 978-1-61208-230-1

For the experiment demonstrated in this publication the software construction activities 'Integration' and 'Transformation' were chosen.

V. THE EXPERIMENTAL SETUP

A. *Technical structure and infrastructure*

The following section utilises this theoretical description to create the basis for the activities of integration, transformation and deployment of real models. The experiment is performed by software engineers using these models. Theses engineers try to perform different transformation and integration software construction activities with and without the support of the proposed solution. The second step comprises a description of the design and implementation of the experiment. These descriptions are intended for the replication of the experiment, and to ensure the sustainability of the experiment for the study's results. The setup of the experiment is divided into three distinct areas:

(1) Description of the environment,
(2) Description of the technical structure of the experiment, the necessary elements, and
(3) Description of the measurement process.

Description of the environment: The experiment was conducted at a German location of the company Schneider Electric (Address Steinheimer Strasse 116, 63500 Seligenstadt, Germany). The company has participated by means of employees at this site and from other international locations using the company intranet. The experiment itself was conducted in normal offices, which provide a connection to this intranet source.

Description of the technical structure of the experiment, the necessary elements: The technical design of the experiment is mainly a hardware and software infrastructure. Figure 4 shows this structure in the environment of the Schneider Electric intranet. Six important elements are involved. The first element is the intranet (1), which is used to connect the various other elements of the technical structure. The second elements (2) are the connected databases, including the software units and complete information about the re-use activities. Four databases are important for the experiment:

1) SOA4D: This is an open source repository software unit with further information about device profiles, including four web services. This repository is based on the Forge technology and offers a web interface.
2) Prometheus SQL: this is a specially developed Repository. It belongs to the approach and uses a Microsoft SQL database and Microsoft SQL database interface.

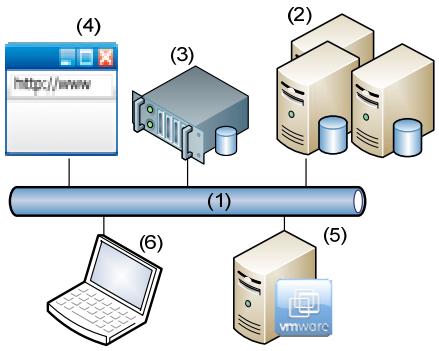

Figure 4. Experimental environment and setup

3) DDXML repos: This is a Schneider Electric internal repository that contains XML elements describing embedded devices. Communication with this repository will be achieved via a Web service.
4) Brick Catalogue: This is, Schneider Electric internal repository used by all Schneider Electric business units containing software unit.

The third element (3) in the experiment's design is the Prometheus Server. This comprises the core of the technical structure. The server maintains information about software units and software construction activities in the connected databases and makes this information available to the user. Finally, the Prometheus Server performs requested activities and presents the available results to users. The fourth element (4) is a website through, which the user can communicate with the Prometheus Server. The website runs on a further server and contains a web application giving the user the ability to query information from the server or to perform reuse activities on the server. This web application is named 'Ecostruxure repository' and for this experiment the 4.1 version was used. The basic technology of the Web application is Microsoft Silverlight version 4.0. The website used the endpoint '/RepositorySearch.html' and was available within the company's intranet. The fifth element (5) of the structure is a VM-Ware server. This server is used to fulfil the experiment's required operating system environment and runs as a virtual machine (VM) to make this available. For the connection to the server VM-Ware Workstation software with version 8.0 was installed on a laptop (6). These elements are common office laptops used within the company Schneider Electric. The laptops were used with the VM-Ware Workstation software with version 8.0. In addition to the computer network environment, there is the possibility to use telephone, internet, voice, conversation, or literature. This is also reflected in the working environment within the company's sites.

 ISBN: 978-1-61208-230-1

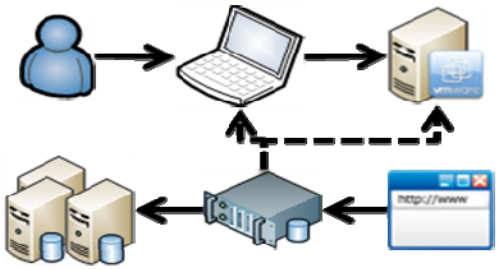

Figure 5. - Basic experiment scenario

Figure 4 shows the scenario based on the experimental setup. Users are able to view the test environment (operating system, the virtual machine) from element (5) (VM-Ware server) by using element (6) (office laptop). Within this test environment, all necessary software applications are found by means searching for information on the Internet, or performing activities on the intranet, as well as various means of communication usually employed by Schneider Electric (FTP, Skype, TELNET). Furthermore, users can now click element (4) (the website) to access and use the Web application, which allows communication with element (3) (Prometheus server). The Prometheus server communicates with the databases that are marked as element (2). Also the Prometheus server interacts with the elements (5) (VM-Ware server) by using element (6) (office laptop) (see Section III). Figure 5 shows this interaction scenario.

Figure 6 shows the different measurement variants in the experimental setup. This can be accomplished by three different (technical) variants. The first is the purely visual recognition of the user's actions and does not require any technical measure (called 'Observer'). The second is to record the user's interactions with the virtual machine as video recording (called 'Recording'). For this, the installed VM Ware Workstation software with version 8.0 is used, which already includes the feature of video recording. The third variant is to log the information (called 'Logging'). This is done in three elements of the experiment's design:

- Create the user data in virtual machines. These data can be analysed after the experiment.
- The Prometheus Server attracts all incoming server requests and performed activities. This information can also be queried after the end of the experiment and used for analysis.
- The data and information are generated and stored in the databases through the interaction of the user.

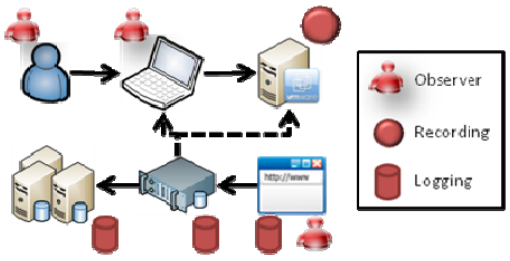

Figure 6. Overview measurement utilities

Description of the technical setup for the measurement and the measurement process itself:

(1) Experimental groups and scenarios: There are a total of three experimental groups: the first group (1) consists of experts for one particular software unit. These individuals receive expert status either because they have created this software unit or are well acquainted with its use. The selection of experts is performed via the Internet from public data of Schneider Electric software units. These data also contain the contact person responsible for this software unit. These people are also asked directly whether they have created the software unit and / or have used it frequently. Altogether the study requires 5 experts. The second experimental group (2) consists of 10 software engineers with the following characteristics: first, the people should actively participate in the software development of a project at the time the experiment takes place. On the other hand, it is important that these people do not have the same expert status as the previously selected 5. The last criterion is that these people are neither expert in the software unit nor in the technology standard development platform for this unit.

The third group (3) is similar to the second experimental group and consist of 10 participants. Therefore, the same rules used for selection of the second experimental group apply.

Note: In this the next phase of the experiment, the total number of participants will be increased up to 30 per group.

Procedure: In principle, there are 3 different experimental groups required to perform seven scenarios. Table 1 shows the different scenarios related to the different groups.

TABLE I. SCENARIOS OF THE EXPERIMENT

Scenario	Description / (GroupID)
(1) Observation of experts	The experts from experimental group (1) performs transformation and / or integration activities (manually). / **(1)**
(2) Collection of software units and activities	Collection of software units and activities: In this scenario, each of the selected experts from experimental group (1) insert the knowledge about the unit and the specific transformation and, or integration activity into the Prometheus Server./ **(1)**
(3) Prometheus Validation	The experts perform the same activities as in scenario (1) but now with Prometheus Server support. The expert validates the results. / **(1)**
(4) Reuse activities with Prometheus	Participants from the group (2) are asked to take over one transformation and integration task. They have to use the Prometheus Server for this purpose. / **(2)**
(5) Reuse activities without Prometheus	In this scenario, the people placed in the experimental group (3) are asked to take over a transformation or integration task. Activities are repeated so they correspond to those of the experts from scenario (1), The Prometheus Server is not

 ISBN: 978-1-61208-230-1

	used / **(3)**
(6)/(7) Validation of the results	Validation of the results: This scenario will test the results of the experimental group (2) and (3) by the experts for the respective software unit from experimental group (1) and (2). / **(1)**

(2) Measurement

In the following section, the methodology of measurement of the experiment will be explained. This includes the definition of the measurable variables and the process of measuring.

Definition of variables: The results of the measurement procedures are stored in the form of variables. In addition, each variable is assigned a unique name within the experiment. In this section, all variables are named and briefly presented. Table 2 shows the different measurable variables in the different scenarios.

TABLE II. OVERVIEW OF VARIABLES

Sc. ID / ID	Name: Description
(1,3,4, 5)/ (A)	**ActivityDuration**: How long does it take an expert/user to perform an activity? This variable contains a value that expresses how long the expert takes for the preservation of the task.
(1,3,4, 5)/(B)	**TaskAnalysisActivityDuration**: How long did it take the expert/user to analyse the task initials? This variable describes the time between being presented with the task and the start of work on the computer.
(1,3,4, 5)/(C)	**TaskActivityDuration**: How much time does expert/user spend working on the computer in order to perform the activity? This variable describes the time between the start and completion of work on the computer activity.
(1,3,4, 5)/(D)	**ActivityCarriedOutSuccessfully**: Has the expert/user completed the activity successfully? This variable represents whether an activity was successful or not.
(1,3,4, 5)/(E)	**UseKnowledgeSources**: What kind of knowledge sources did the expert/user use to perform the activity? This variable describes the sources consulted to perform the activity such as the Google phone or contacting another expert for information.
(1,3,4, 5)/(F)	**MadeSubTasks**: What sub tasks did the expert undertake in order to perform an activity?
(2)/(G)	**EnterUnitDuration**: How long does it take the user to enter all necessary information about a software unit into the Prometheus system? This variable contains a value of the expert testimony of how much time was needed from commencing work on the computer to enter the information of its software unit.
(2)/(H)	**EnterActivityDuration**: How long does the expert take to enter an activity for a software unit in the Prometheus system? This variable contains a value of the experts' statement of how long since commencing work on the computer it took to input the specific activity of entering the activities information.
(2)/(I)	**TotalInputDuration**: How long does it take the expert to enter all the information into the Prometheus system? This variable contains a value of expert testimony on how long the whole process of entering all their data took.
(2)/(J)	**SuccessfulEntry**: Could the expert enter all the important information? This variable tells us whether an expert could enter all the information about a software module and complete activities in the system.
(2)/(K)	**MadeSubTasks**: What sub tasks did the expert undertake in order to perform an activity?
(3,6,7)/ (L)	**ResultIsValid**: Is the result of an activity conducted by Prometheus or without equivalent to the result of the same activity conducted by an expert? This variable indicates whether the expert considers the result of activities performed by Prometheus or without it as good as the result, which was achieved through manual execution of the same activity.

Measurement Execution Process: In Figure 6, three variants of measurement used to measure the variables were introduced. The following section shows, which of these techniques are used for the different variables.

In Scenarios (1), (3), (4), and (5), seven measurements are raised per cycle: (A) The variable 'ActivityDuration' is measured by the observer (measurement variant 1). Here, the observer measures from the time, which he assigns the task to the expert/user up to the time the expert says the task was completed. The time is recorded in whole minutes. (B) The variable 'TaskAnalysisActivityDuration' is determined by the interaction of measurement variant (1) and (2). Here, the observer notes the time at which the task is assigned to the expert/user (see variable 'ActivityDuration'). The end of this phase can be measured at the time when the expert commences an activity on the virtual machine. The time is recorded in whole minutes. (C) The variable activity of 'TaskActivityDuration' determines the interaction of the measurement variants (2) and (1). The point in time at which the activity is started on the virtual machine is measured. The endpoint is the time the expert/user tells the observer that the task was completed. The time is recorded in whole minutes. (E) The variable 'UseKnowledgeSources' is determined by the measurement variants (1) and (2). The observer notes all information coming from the expert's behaviour that cannot be measured by measurement variant (2). The type of measurement (2) also used to analyse, which sources of information accessed through the use of the virtual machine. Typically such sources can be classified by using source names and the type of resource, e.g., (1) co-worker, telephone, and (2) website, Google (Web browser). (D) The variable 'ActivityCarriedOutSuccessfully' is measured by measurement variant (1). The expert/user is asked after the completion of the activity if he has done this successfully. The variable can only be set to yes or no. (F) The variable 'MadeSubTasks' is determined by the measurement variants (1) and (2). Here, the observer notes the progress of the entire task. This can be done based on the recording of the activities in the virtual machine itself, which is operated by the observer both on the external (outside the virtual machine) and internal (within the virtual machine) view. The observer here notes, which activities were measurable, including their start and end time, e.g., starts 10:41 expert uses web browser.

In scenario (2), five measurements are made: (G) The variable input 'EnterUnitDuration' determines the measurement variants (1) and (3). The website (see Figure

4) logs every activity of the user. Accordingly, the entry of the website is the start time and represents the initial value used for the measurement. To avoid error, the observer compares measured time with the automatically measured time. The end time is determined by the expert's signal indicating that he/she has to finish the task. The observer notes down this time. Time is measured in whole minutes. (H) The variable 'EnterActivityduration' is measured by the measurement variant (3) on the Prometheus Server (see Figure 4) and the website (see Figure 4). The server and the website recognize the time of a user's request. Each measurement contains the time and the names of tasks, e.g., 10:00:00 user creates a new software unit. (I) The variable 'EnterActivityDuration' is measured by the measurement variant (1). The observer records the start time point at which he/she hands over the task to the experts. The end time is determined by the expert's signal that he/she has finished the task. The observers take note of this point in time. Time is measured in whole minutes. (J) The variable 'SuccessfulEntry' is measured with the measured variants (1) and (3). Firstly, the expert must inform the observer that he/she was able to enter all information into the system. Secondly, the Prometheus server writes all values into the database. The variable can only be set to yes or no. (K) The variable 'MadeSubTasks' is measured in the same way than in Scenario (1,3,4,5)/(F).

In scenarios (4), (6), and (7) one measurement is made: (L) The variable 'ResultIsValid' is captured by the measurement variant (1). The expert examined the results of the performed activity from the scenarios (3), (4), and (5) with the same activity carried out in scenario (1). It tells the observer whether the result has the same value and is usable. The variable can only be set to 'yes' or 'no'.

Definition of Software units and reuse activities: The different scenarios 1-7 are performed in this experiment with the software units shown in Table 3.

TABLE III. USED SOFTWARE UNITS

Name / ID	Description	Tec/ Unit Type / Repository	Integration effort / Transformation effort
DPWS / SU1	Enable devices for WS* profiles	Java / Component / SOA4D	Advanced into Eclipse/Advanced using IKVM
DPWS / SU2	Enable devices for WS* profiles	C++ / Component / SOA4D	Advanced into Visual Studio / None
CWS / SU3	Webservice for data exchange of business units	Soap-C# / Webservice / Prometheus	Normal into Visual Studio / Advanced using SVCUtil
CWS / SU4	Webservice for data exchange of BUs	Java-Android / Class / Prometheus	Advanced into Eclipse / Advanced using Java2SOAP
Code Signing / SU5	Webservice for Code signing	Soap-C# / Webservice / Brick Repos.	Normal into Visual Studio / Normal using SVCUtil

Table 3 shows that five integration and four transformation activities are connected with the five software units. The integration activities typically focus integration of software units on the most common IDEs (Visual Studio and Eclipse). The transformation activities include the transformation of software units on three different transformation tools (IKVM [16], SVCUtil [17] and WSDL2Soap [18]

VI. EXPERIMENT RESULT DISCUSSION

A. *Experiment Results*

The experiment's results were collected in the way described in the previous section. The next step is to discuss these results. First of all, the result of one software unit with a transformation activity will be discussed in more detail. After this analysis, the results of all software units will be summarised and compared. For this purpose, two perspectives were used for analysing the summarised results: Comparing different groups from the perspectives of (1) activity execution and (2) use of knowledge.

1) Detailed result example

One of the measured software unit is the 'Device Profile for WebServices' Java stack, which enables Java based embedded devices to handle mutable WS* Protocols like WebService discovery. The transformation task for this software unit was to use IKVM transformation tool to transform the complete DPWS Java Stack into a C# Stack. This task requires knowledge about the DPWS Java Stack (especially the references of the 20 different JAR Files), the .NET Platform and experience in using IKVM. This scenario was taken from a real development scenario of Schneider Electric in the European research project for industrial automation SOCRADES [19].

Expert scenarios (1-3): Scenario 1: In the first scenario, the Expert was measured by performing this task manually. The main result is that the experts needs 14:23 min.. In Scenario 2 it was measured how long the expert needs to insert the software unit and the transformation activity. The initial creation of the software unit into Prometheus needs 12:06 min. and the transformation needs 38:03 min.. In Scenario 3, the expert was observed by using the Prometheus Server to perform this task. He needs 2:04 min. to perform the task and received a 2:56 min. training into the system (this training will only be necessary once per expert). The expert validated the result as a correct transformation.

Non-expert scenarios (4-5): In Scenario 4, five non-experienced software engineers of the industrial areas of Building, Power and Industry (Automation) did the task without support of the Prometheus Server.

 ISBN: 978-1-61208-230-1

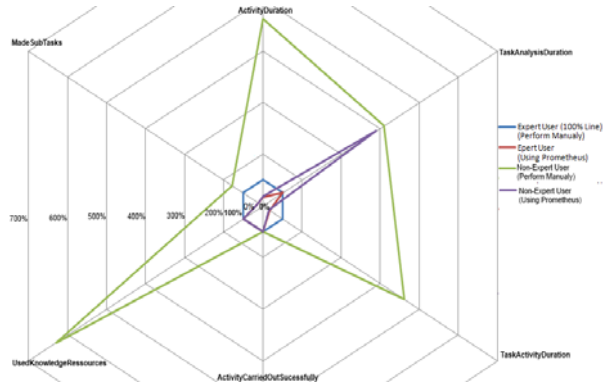

Figure 7. Results of the different groups for DPWS transformation ctivities

The different participants need 42 min., 90 min., 77 min., 69 min., and 104 min. (rounded off). Thus, the average time was 76 min. (rounded off). The expert validates all final results as valid. In Scenario 5 the participants of group (3) use Prometheus to perform the task. The measured introduction task performing times (in minutes) were (3:03/2:23), (2:56/2:10), (2:33/1:59), (2:45/2:22), and (2:43/2:23). The average time was (2:48/2:18). The expert validates the results as correct results. Figure 7 summaries the results. The validation in Scenario 6 and 7 are not shown in Figure 7 because of all results were valid. Additionally to the measured time the kind of used knowledge resources were measured. Only online websites, downloaded documentation, and the expert were used as knowledge resource. The expert in scenario 1 uses only one knowledge resource (an older development project) 4 times. By adding the necessary information into the Prometheus system of Scenario 2 the expert only uses one knowledge resource (the introduction). In Scenario 3, the experts need only the introduction to perform the activity. The non-expert group (2) of scenario 4 needs multiple resources multiple times. Figure 8 shows the used number of knowledge resources in each scenario (average values).

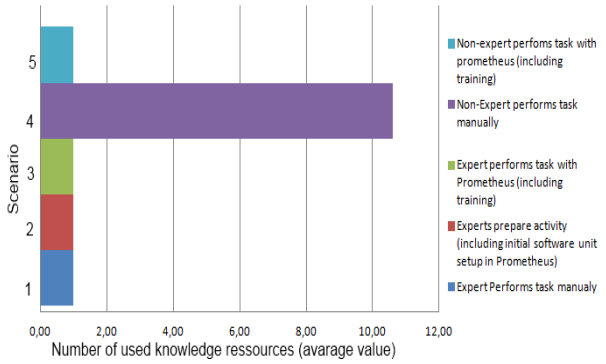

Figure 8. Overview of number of used knowledge resources

The non-expert group (3) of scenario 5 needs only one knowledge resource (the introduction).

2) Comparing of different groups from the perspective of activity execution

Figure 9 and Figure 10 show the results of the three groups in transformation and integration activities measured in the Scenarios 1, 3, 4, and 5. The different results of the software units are summarised by using this type of view. In the context of transformation, Figure 9 demonstrates a clear separation of the different groups. Starting with the Expert Users without Prometheus support (Expert, Scenario 1) as the 100% comparison line, the

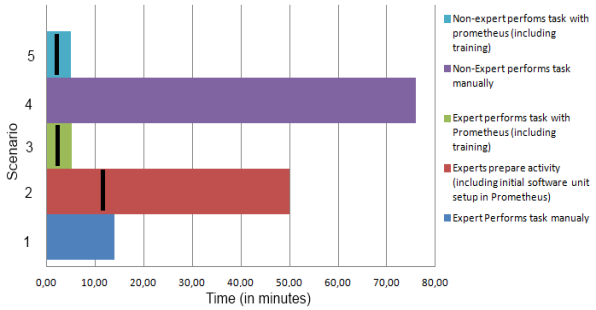

Figure 9. Results of the different groups for transformation activities (5 software units)

measured values of the second group (User with Prometheus support – User (P)) are significantly decreased. This fact is mentioned especially in the variable 'ActivityDuration' (1). On the other hand, the Variable 'TaskAnalysis-ActivityDuration' (2) is much closer to the comparison line. As a result, Prometheus Users are able to perform a specific activity much faster than an expert user or a Non-Expert user. In comparing the two variables of the comparison line with user (without Prometheus support User) Figure 9 shows a further significant difference. Both variables of the user are decreased. The normal user needed much more time to fulfill the given tasks. But this difference changes by analyzing the results of users (with Prometheus support). Compared to the expert with Prometheus support this group has no significant differences, but compared to the expert group without Prometheus support the measured values decrease significantly. In Figure 9, the two lines of Prometheus supported users are more or less congruent.

As a result of this consideration, it is clear that the Prometheus approach creates a positive effect for Non-Expert User and even for expert users.

Figure 10 shows the measured values for the integration activity. The first interesting point is the general comparison to the results shown in Figure 9. Both pictures show nearly the same result, but the positive characteristics are not so distinct. Only the users (without Prometheus support) performing the integration activity need less time (compared to the 100% comparison line) then the same group was

 ISBN: 978-1-61208-230-1

performing the transformation activity. That both results a nearly the same indicates that the used approach supports software engineers by performing these kinds of activities.

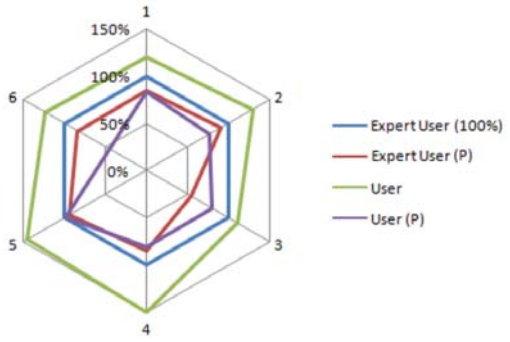

Figure 10. Results of the different groups for integration activities

All users (experts and non-expert user) were able to perform the given activities correctly and needed less time than the expert user (without Prometheus support).

3) Comparing of different groups from the perspective of the use of knowledge

In Figure 9 and Figure 10, it is also mention that most of the expert users (80%) (without Prometheus support) did not use a measurable knowledge base. The other 20% used exactly one knowledge base. All experts or users (with Prometheus support) only used the knowledge base that was the documentation of the Prometheus system. The users (without Prometheus support) performing both the transformation and the integration activity used much more knowledge bases. The most used knowledge base was the internet.

B. Impacts on industrial reuse

In applying the aforementioned approach to industrial environments faced with both creator and reuse phase dilemmas, and therefore no knowledge transfer, leads to the following effect, shown in Figure 11: The effort for the creation team increases by adding the software unit information into the Prometheus system. The theoretical very useful but missing support effort is mostly replaced by the effort for this 'knowledge injection'.

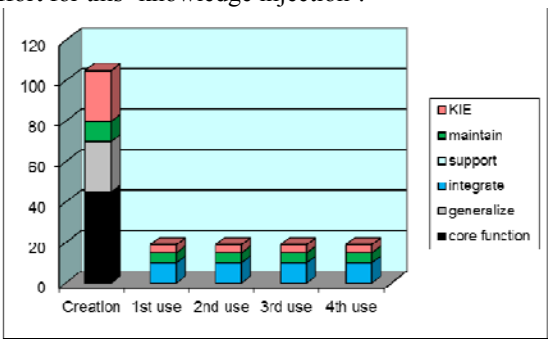

Figure 11. Effects on MTwKIE

The major effect is visible at the reuse site. Even without or just less support, the effort for reuse for single users or team is significantly reduced. In the case of this experiment the reduction of the measured variable are ~38,5% in the transformation activity case compared to the expert user (perform manually) (see Figure 9), ~ 73,21% in the transformation activity case compared to the non-expert user (perform manually) (see Figure 9), ~38,5% in the integration activity case compared to the expert user (perform manually) (see Figure 10), and ~ 73,21% in the integration activity case compared to the non-expert user (perform manually) (see Figure 10). This is mainly based on the fact, that expert and non-expert Prometheus users do not spend much time in searching a software unit and preparation/execute a specific reuse task. The same positive effect is expected in the reuse of a software unit multiple teams of different business units. The approach detailed in this paper has two positive effects. First of all, the solution is sustainable for all teams as it is available to all once it has been stored in the system. This is shown by using different participants from different business units. As consequents, all teams will obtain the same result and the same effects described in Figure 9 and 10. Therefore, the way of reuse planned in the creation phase is more sufficient. The second positive effect is the adaptation towards knowledge created in the "reuse" steps. If a team recognizes an alternative way to perform the reuse activities it is able to store this knowledge in the system. This requires training for the use of the Prometheus system, but other teams are now able to decide, which kind of transformation rule they want to use. (Reuser is Creator) Figure 11 shows both positive effects.

VII. CONCLUSION AND FUTURE WORK

The reuse of a software unit consists of different reuse activities. To perform such activities knowledge is required. Especially in an industrial environment this constitutes problem for a single team and in different teams of different business units. This paper shows the structure and result of an experiment aiming to demonstrate that it is possible to automate chosen reuse activities so that less experienced users are able to perform the activities. By comparing a group of software unit experts, a group of less experienced users within a normal development environment, and a group of less experienced users with the support of the focused automation approach following results are obtained: (1) It is possible to automate reuse activities. Expert users store their knowledge into a system, which is then able to perform the activity (knowledge extraction). (2) Less experienced users who are normally unable to perform such activities are now able to do this. (knowledge injection) (3) Analysing of the results demonstrated that this approach has positive effects for reuse of software units in industrial

 ISBN: 978-1-61208-230-1

environments. (4) With automated support, a single team can decrease their reuse costs from the first time of reuse and thereby make it sustainable. Users utilizing the new approach are able to perform an activity faster than the software unit expert because the system provides the complete environment for the activity based on the expert users' knowledge. (5) By reusing the expert's knowledge, the variations are minimized. All teams use the same activity based on the same knowledge. (6) New automated activities are sustainable because the activity will be changed or a new one is stored in the system, therefore it can be used in each new reuse step of each team. Next to the positive effects, this paper's experiment is limited to two software reuse activities: Transformation and Integration. These activities were chosen because they require different amount of knowledge about tools, environment, and software units. But there also other reuse activities like test, validation, and deployment. Especially for deployment, for example on embedded devices, knowledge is required, but not all activities may be automated completely. The next step is the phase two of the experiment. The number of software units is raised to 10 and the number of inexperienced software engineers in the groups 2 and 3 is increased up.

Next to the fact that the results have to be confirmed by repeating the experiment with new software units and other software engineer the process has to be proofed by other companies. For that purpose the process of the experiment has to be formulated in a formal way. Additionally the following aspects are interesting for the future.

Horizontal extension of the research field: The concept presented in this work was demonstrated by using the example of integration and transformation. But, much more than the activities made use of in this experiment still exist in the area of software unit reuse. First, standard activities exist such as testing and validation of interfaces. These activities usually have a high degree of automation. However, these approaches are lacking in one approach, which is used to represent knowledge uniformly and then re-applied to the different existing automation systems. The scientific task is thus to consider whether the approach presented in this work can also be used for other horizontal activities. On the other hand, technological progress can ensure new activities in the area of reuse. The scientific problem in this case is to check whether the approach presented in this work is can also be used for new activities.

VIII. REFERENCES

[1] I. Sommerville, Software engineering, Pearson, 2011.

[2] F. Bobillo, M. Delgado, and J. Gómez-Romero, "Representation of context-dependant knowledge in ontologies: A model and an application," Expert Systems with Applications, vol. 35, no. 4, pp. 1899–1908, 2008.

[3] N. Juristo and A. M. Moreno, "Reliable knowledge for software development," IEEE Software, vol. 19, no. 5, pp. 98–99, 2002.

[4] R. Oliveto, G. Antoniol, A. Marcus, and J. Hayes, "Software Artefact Traceability: the Never-Ending Challenge,", pp. 485–488, 2007.

[5] M. Zinn, "Service based software construction process," in Proceedings of the Third Collaborative, Plymouth, UK, pp. 169–184, 2007.

[6] M. Zinn, G. Turetschek, and A. D. Phippen, "Definition of software construction artefacts for software construction," in In proceedings of the, pp. 79–91, 2008.

[7] J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," Journal of Systems and Software, vol. 83, no. 1, pp. 67–76, 2010.

[8] V. C. Garcia, E. S. de Almeida, L. B. Lisboa, A. C. Martins, S. R. L. Meira, D. Lucredio, and R. P. de M. Fortes, "Toward a Code Search Engine Based on the State-of-Art and Practice,", 13th Asia Pacific Software Engineering Conference (APSEC'06), Bangalore, India, pp. 61–70, 2006

[9] T. Mens and P. Vangorp, "A Taxonomy of Model Transformation," Electronic Notes in Theoretical Computer Science, vol. 152, pp. 125–142, 2006.

[10] R. Stair and G. Raynolds, Principles of information systems, 10th ed. Boston Mass.: Course Technology Cengage Learning, 2011.

[11] T. Davenport, Working knowledge : how organizations manage what they know, Harvard Business School Press, 2000.

[12] A. Kleppe, MDA explained : the model driven architecture : practice and promise., Addison-Wesley, 2003.

[13] K. Czarnecki, Generative programming : methods, tools, and applications., Addison Wesley, 2000.

[14] M. Zinn, K. P. Fischer-Hellmann, and R. Schoop, "Reuseable Software Unit Knowledge for Device Deployment," presented at the Entwurf komplexer Automatisierungssysteme (EKA 2012), 2012.

[15] M. Zinn, K. P. Fischer-Hellmann. "Reusable Software Units Integration Knowledge in a Distributed Development Environment," International Workshop on Software Knowledge (SKY'11), pp. 24–35, 2011.

[16] J. Frijters, "IKVM," IKVM.NET Home Page, [Online], http://www.ikvm.net/. [retrieved: 09,2012].

[17] Microsoft, "ServiceModel Metadata Utility-Tool,"[Online], http://msdn.microsoft.com, [retrieved: 09,2012].

[18] Apache, "WebServices - Axis." [Online]. http://ws.apache.org /axis/java/user-guide.html, [retrieved: 09,2012].

[19] Socrades, "Socrades Website", [Online], http://www.socr ades.org [retrieved: 09,2012].

 ISBN: 978-1-61208-230-1