INFORMATICA, 2008, Vol. 19, No. 3, 321–344
© 2008 *Institute of Mathematics and Informatics, Vilnius*

A Framework and Tool-Support for Reengineering Software Development Methods

Marko BAJEC, Damjan VAVPOTIČ
University of Ljubljana, Faculty of Computer and Information Science
Trzaska 25, 1000 Ljubljana, Slovenia
e-mail: marko.bajec@fri.uni-lj.si, damjan.vavpotic@fri.uni-lj.si

Received: 5 July 2007; accepted: 11 October 2007

Abstract. The purpose of the research described in this paper is to propose a framework and supporting tools that will help software companies to establish formalised methods that will be technically and socially sound with their needs. Following the framework the companies can asses and improve their existing ways of working, capture them into formalised methods and continuously enrich them based on the past development experiences. Furthermore, the formalised methods that are designed based on the suggested framework are flexible and can be automatically adjusted by the supporting tools to suite circumstances of a particular project or team. This paper describes the framework philosophy and its tool support.

Keywords: software process, software process improvement, software development method, situational method engineering, situation factors and suitability, method adaptation and extension techniques, computer aided method engineering (came) tools.

1. Introduction

In the software development arena both, practitioners and researchers known that software development methods are not followed rigorously as one would expect according to the literature and theory. It has been empirically shown that in practice methods are underused and that their use in not increasing (see, e.g., Hardy *et al.*, 1995; Huisman and Iivari, 2002; Huisman and Iivari, 2003; Fitzgerald, 1998; Middleton, 1999). This causes several problems related to software development process and its results. One of the problems that we specifically focus on in this paper is inability to learn from past development experiences. This happens if the development process is based on *ad-hoc* approaches that tend to be applied on an individual or case-by-case basis and if the knowledge and *know-how* gained through project performance is not captured and disseminated among the developers. Furthermore, several commercial methods that are still in use today are found to be *inflexible* (i.e., they do not permit virtually any adjustment to be made to suite specific circumstances) and are consequently found useless and are thus refused by the development teams (see, e.g., Fitzgerald, 1998; Middleton, 1999).

In this paper we present a framework and tool support for reengineering software development methods which we have developed to address the aforementioned problems.

The framework turns to be very useful for companies that wish to improve their software processes by establishing formalised methods that are sound with their real needs and allow for adjustments to suite specific circumstances. The contribution of our work should be recognised in the robustness and applicability of the framework[1].

The paper is organised as follows. In Section 2 we describe the research approach that we adopted for the needs of our work. Next is the related works section that briefly describes related research areas and explains how our work fits into this research. The core of the paper is then presented in Section 4, where the philosophy and main components of the suggested framework are described, and in Section 5 that presents the toolset that was developed in support of the framework. The paper ends with concluding remarks and ideas for further work on the subject.

2. Research Method

The MasterProc project was organized as a *collaborative practice research* (Mathiassen, 2002) using a combination of *action research*, *experiments* and *study practices*. *Interviews* and *surveys* were used to carry out the assessment of the existing state of the art of software development methods in each of the participating software companies. The main focus of the assessment was to determine how socio-technically suitable are the methods for typical projects carried out by each of the software companies. Furthermore, the goal was to identify the level of flexibility of the existing processes. The information that we received from the interviews and surveys was complemented by action research. For each of the participating software companies a working team was set up comprising two researchers and two practitioners. The main responsibility of the team was to take part in real projects to get firsthand information. The practitioners acted as project managers and methodologists, while the researchers were more or less observers.

In the organization of the MasterProc project the principles of a general *learning cycle* have been adopted, i.e., interpret current situation, find ways to improve practice, plan and implement improvements, and learn from the actions taken. The CPR supports such learning cycle by the three goals it identifies: to understand the current state of software development, to build new knowledge that can support practice, and finally to plan changes and implement them as necessary. After implementing the improvements, the interpretation of the lessons learned have to take place, hopefully leading into the next learning cycle.

3. Related Works

The main principles on which we build our research can be found in two autonomous but related research areas: *Software process improvement* (SPI) and *Situational method*

[1]The framework has been developed under the MasterProc project. The project was co-founded by the Slovenian Ministry of Higher Education, Science and Technology, European Commission and the participating Software Companies.)

engineering (SME). While the main purpose of the SPI is to facilitate the identification and application of changes to the software development process in order to improve the product, the SME primarily deals with developing or tailoring software methods in order to facilitate specific projects and circumstances. The introduction of a specific SME approach into a software company to improve the flexibility of its existing methods can be thus seen as a specific step towards SPI. In this section we shortly describe both research fields and their relation to our work.

3.1. *Software Process Improvement*

Today, many organisations are trying to adopt models of *total quality management* (TQM) principles. In the software development arena these efforts typically manifest through software process improvement (SPI) initiatives of software companies that strive to improve the quality, safety, and reliability of the software they develop and in this way try to increase productivity and customer satisfaction with their products.

One of the commonly known models in the SPI is the *capability maturity model* (CMM) or more recent *capability maturity model integration* (CMMI), which represents a central framework for software quality and process improvement (see, e.g., Paulk *et al.*, 1993; Pressman, 2004). The CMM introduces five levels of maturity into which an organisation can fall according to the quality of their software processes. The five levels are: *initial*, *repeatable*, *defined*, *managed* and o*ptimised*. While in the initial level (level 1) the process is typically ad-hoc and chaotic, the repeatable level (level 2) introduces basic project management processes to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications. In level 3 (defined), the software process for both management and engineering activities is documented, standardized and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software. In level 4 (managed), detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled. Finally, in level 5 (optimized), continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies[2].

In our framework we use CMM as a model against which we evaluate how mature are specific software processes and identify desired maturity levels, i.e., the maturity levels the evaluated organisations want to achieve. Building on the empirical studies that have shown there is a correlation between CMM levels and software quality (Harter *et al.*, 2000; Parzinger and Nath, 2000), we assume the increased maturity will lead also to the improved software quality. The use of the framework for method reengineering inherently leads to at least level 3 (defined) while it includes also activities, such as constant measurement of success and continuous evaluation and feedback from the process that can lead to higher levels of CMM maturity, i.e., level 4 (managed) and level 5 (optimised).

[2]The description of CMM maturity levels is based on Paulk *et al.* (1993).

Another part of our framework in which SPI plays an important role is *method improvement*. Different studies argue (Niazi *et al.*, 2006; Herbsleb and Goldenson, 1996) that managers require more information on how to implement an innovation in order for the implementation to be successful. An interesting model has been proposed by Niazi *et al.* (2006) that measures the maturity of an organisation for the implementation of SPI. It defines different success factors and barriers that are critical for implementation of SPI. In our framework we use these factors to create most efficient scenarios for improvement of method.

3.2. *Situational Method Engineering*

As described above, if we want to achieve the maturity level 3 or more, all projects must be performed according to an approved, tailored version of the organization's standard software process for developing and maintaining software. This is where SME fits in. In the SME literature, a number of approaches can be found that propose how to create project-specific methods. One that is probably the most popular is based on the so-called *reuse strategy*. In this approach a new method is constructed from the fragments of existing methods. The notion of method fragment was introduced by Harmsen *et al.* (1994) who defined it as a reusable part of a method. Fragments can be further categorized into product and process fragments depending on the perspective they cover. Much effort has been put into decomposing existing methods into fragments (Brinkkemper *et al.*, 1996). Also, different repositories have been proposed for their storage (e.g., Harmsen *et al.*, 1994; Brinkkemper *et al.*, 1996; Ralyté *et al.*, 2003). The method construction using the reuse strategy is, however, far from easy, as the fragments have to be first retrieved from the repository, changed if necessary and than assembled together into one consistent and congruent method.

Another approach to SME, known from the literature as the *extension-based approach*, uses the *extension strategy*. In this approach, method engineers are provided with extension patterns that help them to identify typical extension situations and provide guidance to perform extensions. In their work, Ralyté *et al.* (2003) describe two possible ways to perform extensions: (a) directly through matching extension patterns stored in a library to satisfy the extension requirements, and (b) indirectly through first selecting a meta-pattern corresponding to the extension domain and then guiding the extension applying the patterns suggested by the meta-pattern. Karlsson and Ågerfalk (2004) have, however, criticized this approach for not considering situations that are actually very frequent in practice, i.e., when a method is both extended in some fragments and reduced in others. As a solution they proposed a new method for SME that uses a combination of the cancellation and extension operators. They named it *method for method configuration* (MMC). The MMC differs from the aforementioned approaches also in the fact that it does not deal with modular construction of a method but rather with method tailoring taking a particular method as the starting point. From the literature, it is clear that this approach has been somewhat overlooked by the method engineering research in the past.

Finally, the approach to SME that seems to be a result of the most recent efforts in the method engineering research is the *paradigm-based approach* (Ralyté *et al.*, 2003) a.k.a.

evolution-based approach (Ayed, 2004). This approach is founded on the idea that the new method can be obtained either by abstracting from an existing model or by instantiating a metamodel. A new method is then created by first constructing a product model and then process model while for the construction of both product and process model different strategies are available.

For the purpose of our framework we created our own approach to SME which uses a combination of the meta-modelling and extension/reduction based approaches. The approach shares several commonalities with other approach to SME, but most notably with MMC. Both, our approach and MMC suggest configuring an existing method rather then assembling fragments from different methods to construct a new one. Detailed description as well as comparison between our approach and other SME approaches can be found in Bajec *et al.* (2007).

4. A Framework for Method Reengineering

The idea that lies behind the framework for reengineering software development methods is relatively simple. It is based on the assumption that in each software development company, patterns of work could be found that tell how the company is developing software. While a large percentage of software companies own some kind of formalized methods (typically commercial methods), empirical investigations show that what they really do on IT projects differs a lot from what is written in the methods they own (e.g., Fitzgerald, 1998; Bajec *et al.*, 2004). Our assumption in the suggested framework is that in a typical software company the ways of working are sufficiently repeatable to be captured into a formalized method (*base method*) reflecting how the company actually performs its IT projects. If base methods are captured and represented in the way we suggest in this paper then project-specific methods can be created on-the-fly almost without any need for method engineers to intervene. This is done by processing the rules that define, for each method component, in what circumstances its use is *compulsory*, *advisable* or *discouraged*. The configuration process is however interactive. The questions that are subjective in their nature and influenced by particular developers involved in the project can be addressed when they arise and users may intervene as they wish.

The framework consists of four distinct but related phases: (I) *Method Construction*, (II) *Method Configuration*, (III) *Method Use* and (IV) *Method Evaluation and Improvement*. In the remaining part of this section each of the phases will be described in more detail.

4.1. *Method Construction*

Method construction is probably the most important phase of the method reengineering framework and a prerequisite for the other phases. Its aim is to construct a base method that will provide formal description of how the organization that is being analyzed is performing its project. Furthermore, the construction of a base method is crucial as it presents a foundation for creating project-specific methods on-the-fly. Due to the limits of space we will provide here only a brief description of the main activities of the method construction process. For details please refer to Bajec *et al.* (2004) and Bajec *et al.* (2007).

The construction of a base method is a process that has to be done for each organization individually. It starts with the analysis of existing practice in the company and leads into identification of the parts that are technically and socially sound and those that are in these respects problematical. For the analysis of the socio-technical suitability of the existing practice an evaluation model has been designed that facilitates the evaluation (Vavpotič *et al.*, 2004). Possible improvements to the existing practice are then suggested and discussed with the company's development team. Once the vision for the new method is developed and accepted, a *metamodel* is designed that helps to formalize the method. The metamodel can be developed either from scratch or from existing metamodels that have been recently constructed to both underpin and to help formalize methods. Those represent a good source for selecting generic concepts for method formalization. Finally, the metamodel is instantiated and fragments of the base method are captured. Besides the fragments of the existing practice that have been previously approved as technically and social appropriate, many new fragments may emerge. These are based on the suggestions for improvements that have been identified within the analysis of the existing practice. The fragments are first classified according to the underlying metamodel and then described using templates. The templates, which belong to the metamodel, outline how elements of a certain metamodel type should be described.

For the purpose of representing a base method we designed a generic data structure that can be used to underpin any metamodel. The idea of a generic data structure is to allow method engineers to design metamodels according to their perception of how their methods should be formally represented.

Fig. 1 illustrates the main components of the aforementioned *generic data structure*, *base method* and *project-specific method*. The classes representing metamodel are: a

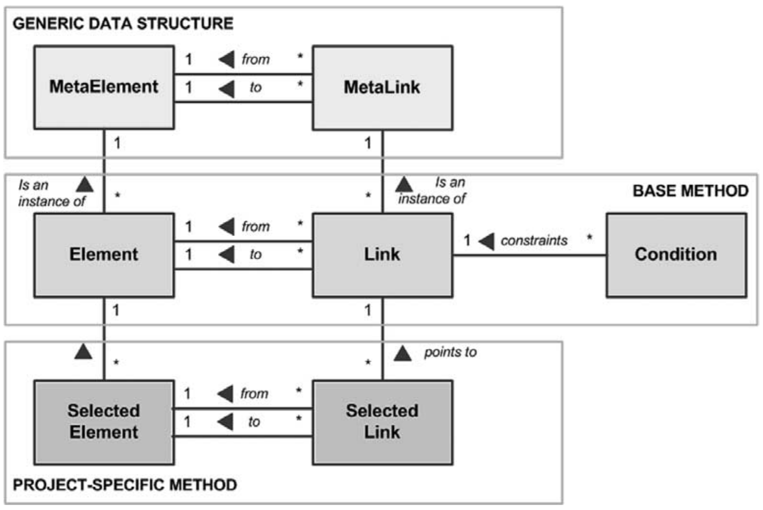


Fig. 1. A generic data structure.

metaelement (it can be of two types: *content element*, such as activity, tool, discipline, role, etc. or *process flow element*, such as decision node, join and synchronization) and *metalink* (links between metaelements). By using such a generic data structure, a base method is represented as a structure of instances of the metaelements and metalinks, and a project-specific method is represented as a selection of the elements and links of the base method.

As mentioned before, a base method encompasses various situations that may occur when projects are performed. In other words, it comprises a number of elements and their alternatives which describe several possible ways to perform a particular project (similar to *project paths* as defined by Hares (Brinkkemper *et al.*, 1996). The paths and method structure, however, are not static. They are defined by the rules that tell which elements to consider in specific circumstances and consequently which path to take. As depicted in Fig. 1, rules apply directly to the links that bind elements of the method (see the element *Condition*).

Besides the rules that put constraints on the links between elements of the method there are also other types of rules that play important role in the suggested framework. In general, they can be categorised into *constraint rules* and *facts*. Since in configuring the base method for the needs of a particular project or situation these rules play essential role we will explain their taxonomy in more detail.

4.1.1. *Constraint Rules*

Constraint rules can be seen as assertions that constrain some aspect of the procedure for constructing project-specific methods. They can be decomposed into four subgroups: *process flow rules*, *structure rules*, *completeness rules*, and *consistency rules*.

Process flow rules are rules that define conditional transitions among activities in the process view of a method. They define the conditions that have to be met to perform a particular transition. For example, in Fig. 2., the rule R_1 defines a conditional transition to the activity *Analyse Logical Structure* while the rule R_2 determines in what circumstances the activity *Analyse Logical Structure* can be omitted.

Similar to process flow rules are rules that belong to the structure rule category. Their distinction is that they can constrain any link between method elements and not just links between activities. In Fig. 2, the rule R_4 represents an example of a structure rule. It constrains the link between the activity *Develop Prototype of the System* and the tool *MS Visio*.

Structure and process flow rules that belong to a base method of a particular organisation actually define *project characteristics* that are important at a particular stage of projects performed by the organisation. Examples of process flow rules (rules R_1, R_2 and R_3) and structure rules (rule R_4 and R_5) are provided below[3].

- R_1: If the process is in the decision node 1 and the *scope of the system* is *large* or *incremental SDLC* is chosen then go to the activity *Analyse logical structure of the system*.

[3]The rules are here written in natural language to ensure their understanding.

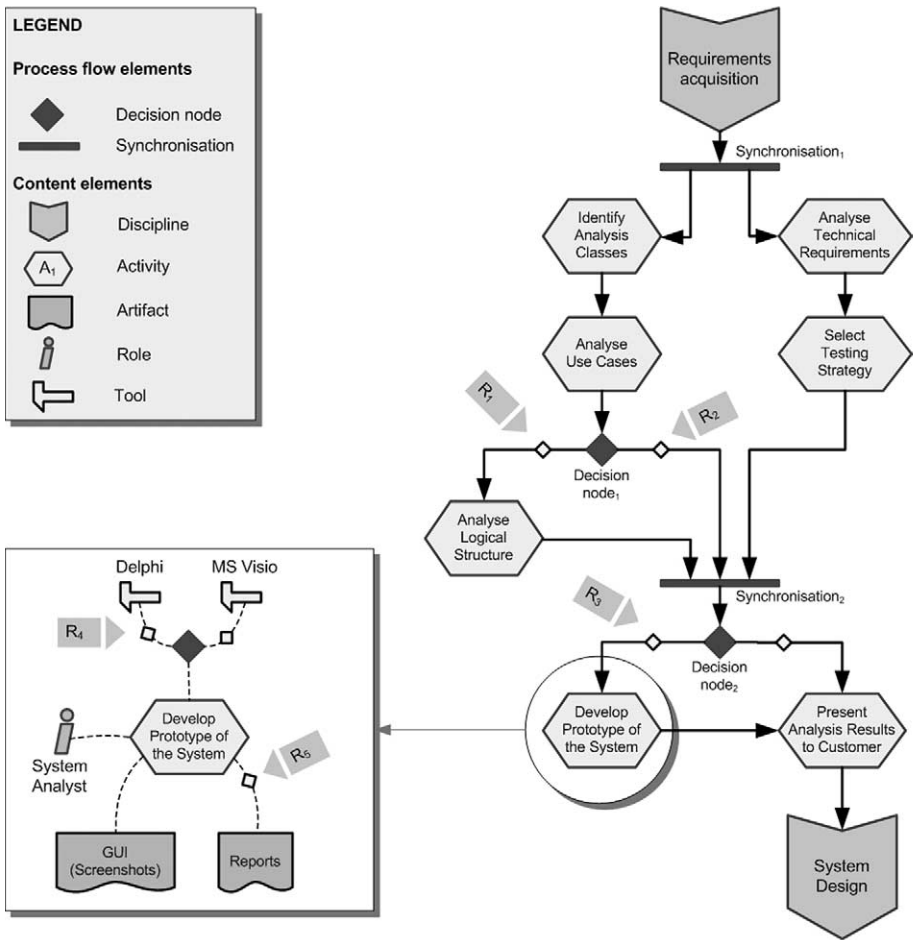


Fig. 2. Representation of a base method.

- R_2: if the process is in the decision node 1 and the *scope of the system* is not *large* and *incremental SDLC* is not chosen then go to the synchronisation point 2.;
- R_3: if the process is in the decision node 2 and the *problem domain* is *new* or *customer requires the prototype of the system* then go to the activity *Develop prototype of the system*;
- R_4: if the process is in the activity *Develop prototype* of the system and the *time frame for producing the prototype* is *more than 1 month* then develop the prototype of the system using *Delphi tool*;
- R_5: if the process is in the activity *Develop prototype of the system* and *important reports are to be developed* then create output artifact *Reports* as a part of the prototype.

Project characteristics, such as *project length*, *project risk*, *project complexity*, *the scope of the system*, *the number of parties involved*, etc. and their respective domains are defined within the organisation's base method. However the values that these characteristics receive are project-specific and are thus defined during the configuration process.

Besides process flow rules and structure rules that both put constraints on associations between elements of a base method the constraint rule category comprises also *completeness* and *consistency rules*. The purpose of these two subcategories is to assure that each project-specific method, created from the elements of a base method, is complete and consistent.

Completeness rules apply – in contrast to the process flow rules and structure rules – to a metamodel and not to a base method (see Fig. 1). Their responsibility is to define the conditions that must be met when creating a project-specific method. Completeness rules actually help to check whether a project-specific method that has been created includes all required components. For example, an organisation may decide the following rules have to be followed when creating methods for projects:

- R_6: each *activity* except the last one must have at least one *successor activity*;
- R_7: each *activity* must be linked with exactly one *role*;
- R_8: each *technique* must be linked with at least one *tool*, etc.

Consistency rules are the last category in the group of constraints. They are similar to completeness rules. Their goal is to assure that the selection of fragments comprising a project-specific method is consistent. While completeness rules only apply to elements that are linked together, consistency rules deal with interdependency between any two elements. In other words, for each element e they determine a set of other elements E that need to be included into a project-specific method if e is included. In the example below the rule R_9 asserts that *the deliverable Business model is dependent on the activity Business modelling*.

- R_9: the deliverable *Business Model* depends on the activity *Business modelling*.

This means that if the deliverable Business model is selected for the inclusion into a project-specific method, the activity Business modelling has to be selected too. While such a dependency may seem trivial it is important as it helps to avoid conflicting situations.

4.1.2. *Facts*

Another important group of rules that are considered during the configuration process are facts. Facts are assertions that define characteristics of the project for which we create a project-specific method. Depending on how they define project characteristics they can be classified into *base facts* or *derived facts*. Base facts define project variables directly while derived facts are derived from base facts using inferences or calculations. In the examples below, the rule R_{10} is a base fact while the rule R_{11} is a derived fact.

- R_{10}: the *project domain* is *well known*;
- R_{11}: if the *project field* is *telecommunications or healthcare* then the *project domain* is *well known*.

In the method configuration process facts are very important as they are checked when structure and process flow rules are processed. For example, a structure rule might state that "when performing requirements validation there is no need to produce a prototype if the problem domain is well known". To be able to perform this rule we must first check the facts about the project domain to find out whether the domain is well known or not.

As indicated in the examples of the constraint rule category (see, e.g., rules R_3 or R_5) facts can describe virtually any condition that is important for the project. Furthermore, they are created dynamically during the method configuration process. For example, when an element e is selected to be included into a project-specific method this becomes a fact (*e is selected*) which could become important latter on in the method configuration process. Some additional information on rule categiries, their storage and application to IS development can be found, e.g., in Kapocius and Butleris (2005) or Bajec and Krisper (2005).

4.2. *Method Configuration and Use*

Once a base method has been successfully established and discussed with its users it is ready for use. However before it is actually applied to a specific project or situation it has to be configured so that it includes only the components that are relevant to the situation in question. At this point the representation of a base method that was described before reveals its value. With an appropriate tool the adjustment can be done automatically. In this section we describe the algorithms that facilitate the auto-adjustment process.

The algorithm that supports the method configuration process is relatively simple. It starts with an element in the base method (typically this would be a starting activity) and ends when there is no link that would connect the current element further with any other element. If such links are found they are examined for constraints they might have. When a particular link has no constraints or when constraints exist but are satisfied than the element at the end of that link is processed in the same way using recursion.

```
PROCEDURE CreateProjectMethod(pm,e);
// pm - project method, e - starting element of the base method
BEGIN
  Find links for the element e
  For each Link l
    IF conditions are satisfied for the Link l
    THEN
     Mark the output element of the Link l as selected for the pm
    Mark the Link l as selected for the pm
    CreateProjectMethod(output element of the Link l,pm)//recursion
    END IF
  NEXT
  END;
```

When a project-specific method is created using the algorithm above, the elements that have been selected has to be checked for consistency and completeness. The verification algorithms below show how this can be done.

```
PROCEDURE CheckCompletness (pm);
// pm - project method
BEGIN
  //completeness verification
  Select all links from the pm
  For each Link l
    //Check the completeness constraint for the Link l
    Count the links that connect the input element of the Link l
    with the output elements of the same type as is the output
    element of the Link l
    IF the number of links is outside the min, max limits
    THEN mark the Link l as problematical.
  NEXT;
END;

PROCEDURE CheckConsistency (pm, e);
// pm - project method, e - starting element or
// link of the project-specific method
  BEGIN
  //consistency verification
  Select the set of elements and Links D that e is dependent on
  For each element or Link d from D
    IF d is not selected THEN Mark d as problematical
    CheckConsistency(pm, d) //recursion
  NEXT;
  END;
```

For detailed description on the process configuration approach, its comparison with other SME approaches, as well as on the experiences with its application in practice, please see Bajec *et al.* (2007).

4.3. *Method Evaluation and Improvement*

In the suggested framework it is essential that the underlying base method and corresponding rules continuously evolve as a reflection of knowledge and experiences acquired through project performance. This means that when using the framework new fragments may emerge as a result of situations that are specific and thus not yet supported by a current base method. In such cases, additional fragments are captured and circumstances for their use are determined. In practice, it actually takes some time for a base method to become *all-inclusive* in terms of providing guidelines for all kinds of situations that may happen in projects a particular company is performing. This phase, in which the base method rapidly evolves, is called the *learning phase*. It takes place in the first few projects after the framework has been introduced into a company. Eventually however, the base method would become more stable and changes on a large scale less frequent.

For the aforementioned reasons the framework provides specific activities for the continuous method evaluation and improvement. To retain social and technical suitability base methods are regularly evaluated and improved. The evaluation is performed on a level of a single method element, which enables precise identification of less suitable method elements, determination of reasons for their unsuitability and creation of improvements consequentially.

The evaluation activities are based on the *method evaluation model*. Although various method evaluation models have been proposed in the past, they tend to consider either only technical (CMU/SEI-2002-TR-029, 2002; ISO/IEC-15504, 1998; ISO/IEC-FCD-9126-1, 1998) or only social (Rogers, 2003; Ayzen, 1991; Venkatesh and Davis, 2000) dimension of a method. However, such partial evaluation does not provide a complete understanding of method's suitability. Therefore, an *evaluation model* was created that facilitates simultaneous evaluation of method suitability on a *social* and *technical dimension*. The social dimension focuses on method's suitability for social and cultural characteristics of a development team and facilitates determination of the level of method's adoption. The technical dimension considers suitability of a method for technical characteristics of a project and an organization, and helps to determine the level of method's efficiency.

Fig. 3 depicts application of the evaluation model in practice. After an evaluation is completed, all method elements are positioned in a scatter plot diagram that is divided into four quadrants distinguishing between four different types of method elements (regarding their value):

- A *useless method element* is both technically and socially unsuitable. Different reasons for such unsuitability can be identified. For instance, unsuitability can be caused by constant technology change that eventually renders a method element

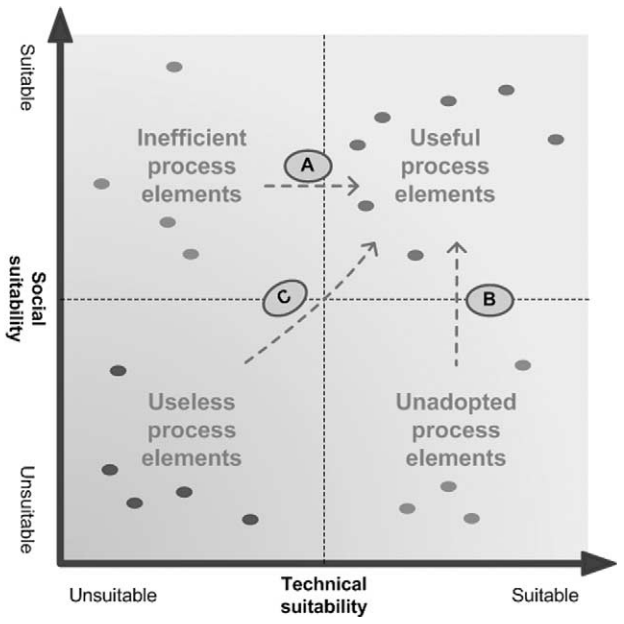


Fig. 3. Application of the evaluation model.

technically unsuitable. Consequently, developers stop using the element, which finally results in its complete unsuitability. Alternatively, an element might have been technically unsuitable from the beginning and therefore never used. There are various methods to measure quality of a specific element (see, e.g., Caplinskas and Gasperovic (2005).
- An *inefficient method element* is socially suitable, but does not suit technical needs of a project or an organisation. For instance, these can be method elements that have been technically suitable in preceding projects and are well adopted among users, but are technically inappropriate for the current project.
- In contrast to an inefficient element, an *unadopted method element* is technically suitable, but its potential users do not use it because it is socially unsuitable. Many reasons why potential users do not adopt a technically efficient method element can be identified. The element might be overwhelmingly complex, it might be difficult to present advantages of its use to the potential users, it might be incompatible with existing user experience and knowledge, etc.
- A *useful method element* is socially and technically suitable. Such method element is adopted among its users and suits technical needs of the project and the organisation.

A method element that is perceived as unsuitable can be improved by using different *improvement scenarios.* The general selection of an improvement scenario depends on the quadrant where the element is positioned. In case of an *inefficient method element* (see Fig. 3, arrow A.), its technical suitability should be improved and social suitability retained. Since users already adopted the element, it should be modified only to the extent that it becomes technically efficient again. In case of an *unadopted but technically suitable method element* (see Fig. 3, arrow B.), the causes for element's rejection among its potential users should be explored. For instance, potential users of the element might lack knowledge and experience to use it. Consequentially the improvement should focus on training of element's potential users rather than on altering the element. In case of a *useless element* (see Fig. 3, arrow C.) that is both socially and technically unsuitable the most reasonable action would be to replace or discard it completely. Most likely a technically and/or socially more suitable element can be found or the element is not needed at all.

Following the general selection of improvement scenarios, limitations of method elements are explored and each improvement scenario is additionally tuned to conform to these limitations (see Fig. 4). We differentiate between two basic types of limitations. The first type is limitations imposed by characteristics of the element itself. A common example of such limitation can be that a method element can only be changed to a limited extent or not at all, due to its interaction with other method elements. In such case, the improvement scenario should focus on additional training of element's users so that they better understand the element and adopt it, but should not modify the element itself. The second type is limitations imposed by characteristics of the development team. For instance, certain development teams are more innovative and therefore more eager to adopt new or changed elements – SPI. Niazi *et al.* (2005) identified critical success

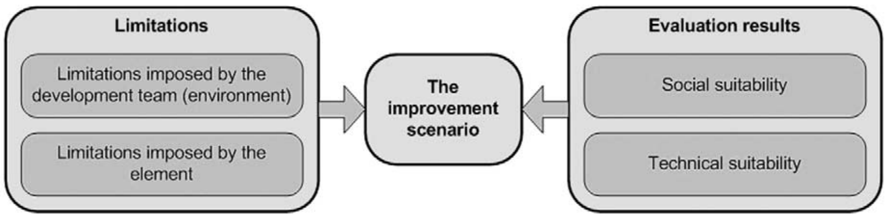


Fig. 4. Formation of an improvement scenario.

factors that influence the adoption of SPI and also consider different aspects of a development team like: team awareness, team experience, management support, team members involvement, time pressure, etc. Based on these critical success factors we additionally adjust the improvement scenario.

After application of the improvement scenarios most method elements are expected to move to the useful method elements quadrant, though some of the elements might still need further improvements or even replacement.

An important aspect of the evaluation is that it is performed by the actual users of the method elements. Each method user evaluates social suitability of the method elements he uses to perform his development activities. For instance, an analyst evaluates all method elements related to analysis, a programmer evaluates all method elements related to programming, etc. In this manner we objectively assess users' attitude towards the method, i.e., social suitability of the method. Evaluation of technical suitability of the method, however, is performed by so called technically advanced method users. They are not only knowledgeable in the development activities they perform, but also understand a broader perspective of the method, are familiar with trends in the field of their expertise and are able to assess technical efficiency of the method objectively. In case, when technically advanced method users cannot be identified for a certain development activities, external experts are engaged to assure objective evaluation of the method's technical efficiency.

Two distinctive qualities of the proposed model can be identified. Firstly, it simultaneously considers social and technical suitability of a method; and secondly, it facilitates evaluation on a scale of a single method element. These allow a software development organization to observe value of its method in detail, to identify technically and/or socially inappropriate parts, and to create customized improvement scenarios based on the evaluation of each method element. For the detailed information on the method evaluation model please see Vavpotič *et al.* (2004, 2006).

5. Tool Support

Creating project-specific methods by tailoring an existing method, assembling parts of different methods together, and evaluating method on a scale of a single method element are all very complex tasks that can not be done without appropriate tool support. Therefore a set of prototype tools, namely Agile Methodology Toolset (AMT), has been

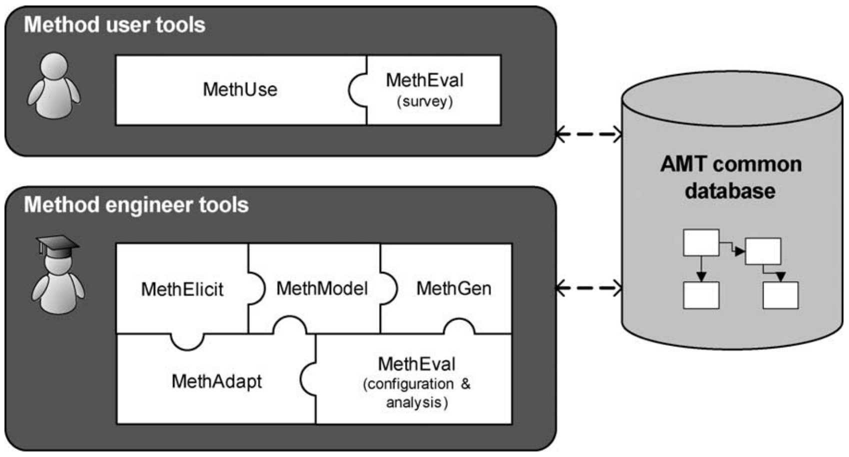


Fig. 5. High level architecture of the AMT toolset.

developed in support of different parts of the suggested framework for method reengineering.

In respect to the classification provided by Kelly (1997), the AMT falls into the group of CAME tools (Computer Aided Method Engineering). Its main purpose is to facilitate the elicitation of the company's base method; create project-specific method by selecting the fragments of the base method that correspond to the project characteristics; and support the method evaluation and reengineering processes.

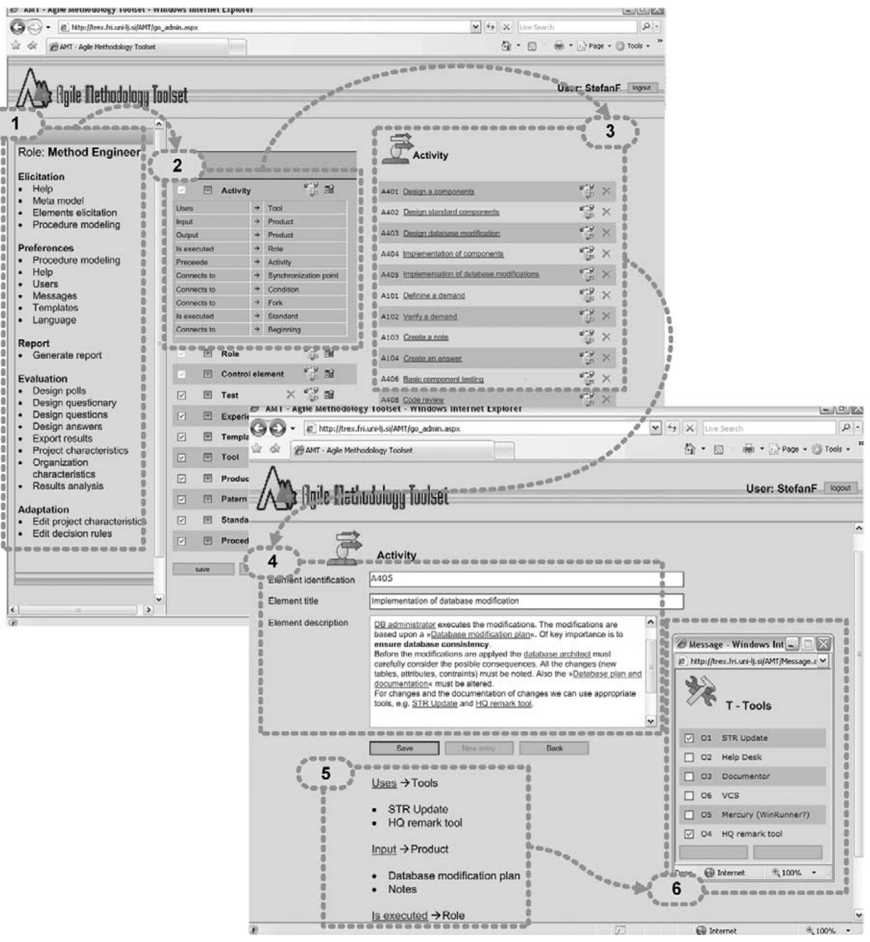
The toolset consists of six interconnected modules (see Fig. 5). Five of the modules focus on the method engineer-side of the method reengineering-related activities: MethAdapt – a module for adapting base method to project characteristics; MethElicit – a module for method elicitation; MethModel – a graphical tool for method modelling; MetEval – a module for method evaluation; and MethGen – a module for generating method reports. Additionally there is a module, which focuses on the use of the method – MethUse, which is also connected to MethEval. The module guides the user through the selected method, which can be the company's base method or the project-specific method. The main integration point for the six modules is the common database where all data and metadata are stored.

In the following subsections the modules are shortly explained.

5.1. *MethElicit and MethModel*

Both MethElicit and MethModel modules are used in the process of method elicitation, during which the company's informal method is formalized (see also Subsection 4.1). The use of MethElicit starts with definition of a metamodel. The method engineer defines the elements of the metamodel and relations between them. The resulting metamodel forms a framework that is filled in with instances of metamodel elements and instances of relations.

Fig. 6 shows the main user interface of MethElicit module and demonstrates a typical sequence of actions during elicitation process. Starting point is the navigator panel (1). The elicitation section of the panel comprises functions for a metamodel editing, elements elicitation and procedure modelling. To define the metamodel, the method engineer selects the first of the three functions. The metamodel editor is displayed (2). It allows the user to add, delete and change metamodel elements, and to define relations between them. The middle part of the upper screenshot shows all possible relations of an element to other metamodel elements. For instance, the *activity* element is related to the *tool* element via the *uses* relation, which means that during execution of a certain activity a certain tool can be used. On the right side of the upper screenshot (3) there is a list of instances of the selected element (in our case instances of the *activity*). Selecting an element instance from the list displays its details in an edit panel (4). The edit panel also displays a list of all instances of relations to other instances of elements (5). Using the



Uses	→	Tool
Input	→	Product
Output	→	Product
Is executed	→	Role
Preceede	→	Activity
Connects to	→	Synchronization point
Connects to	→	Condition
Connects to	→	Fork
Is executed	→	Standard
Connects to	→	Beginning

Activity

A401	Design a components
A402	Design standard components
A403	Design database modification
A404	Implementation of components
A405	Implemenation of database modifications
A101	Definine a demand
A102	Verify a demand
A103	Create a note
A104	Create an answer
A406	Basic component testing
A408	Code review

Activity

Element identification: A405

Element title: Implementation of database modification

Element description: DB administrator executes the modifications. The modifications are based upon a »Database modification plan«. Of key importance is to **ensure database consistency**.
Before the modifications are applyed the database architect must carefully consider the posible consequences. All the changes (new tables, attributes, contraints) must be noted. Also the »Database plan and documentation« must be altered.
For changes and the documentation of changes we can use appropriate tools, e.g. STR Update and HQ remark tool.

Uses → Tools

- STR Update
- HQ remark tool

Input → Product

- Database modification plan
- Notes

Is executed → Role

☑	O1	STR Update
☐	O2	Help Desk
☐	O3	Documentor
☐	O6	VCS
☐	O5	Mercury (WinRunner?)
☑	O4	HQ remark tool

Fig. 6. Main user interface of MethElicit module.

edit panel a method engineer can edit the description of the element instance and define instances of relations. Only instances of those types of relations can be created that have been defined in the metamodel (2). To define an instance of a relation between two element instances, the user must select the appropriate element instance in the popup menu (6). For example, during the execution of the *activity A405* the *STR Update* and the *HQ remark tools* are used.

Using MethElicit module a method engineer can gradually capture all static method elements and their static relations. However, the elicitation of non static method procedure is done by MethModel module. The module not only provides a graphical user interface that enables the method engineer to graphically design a method procedure, but also enables the method engineer to formalize the conditions and rules that are used during method tailoring. To ease the process of building the rules MethModel uses a rules editor that helps the method engineer to enter the rules in a form of if-then clauses (see Subsection 4.1). The technique that is used to graphically represent a method procedure is somewhat simplified activity diagram. It comprises activities, relations, conditions and synchronisations. All of the diagram data is stored in the database as it is also used by MethAdapt module during method tailoring. To display the diagram MethModel creates a bitmap representation of the diagram that can be used for quick previewing of the method procedure (see Fig. 7).

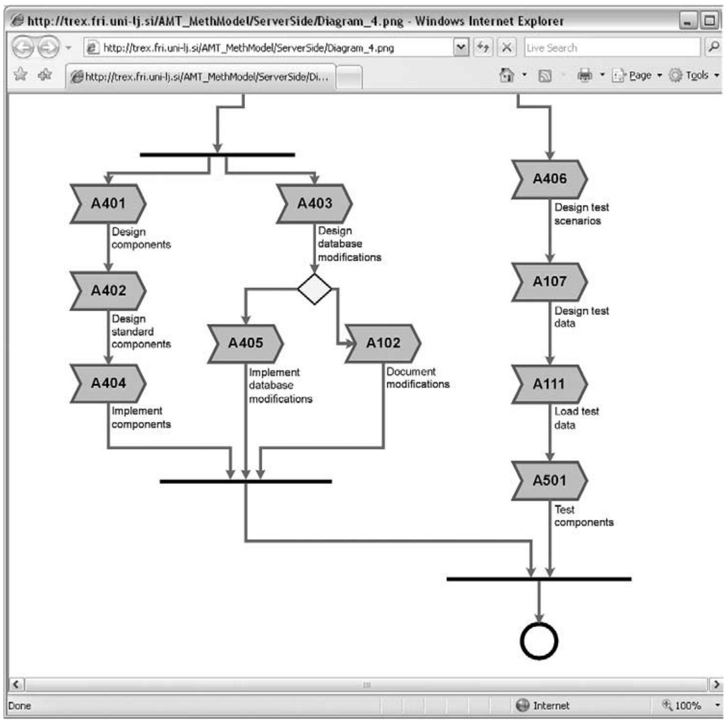


Fig. 7. An activity diagram created using MethModel.

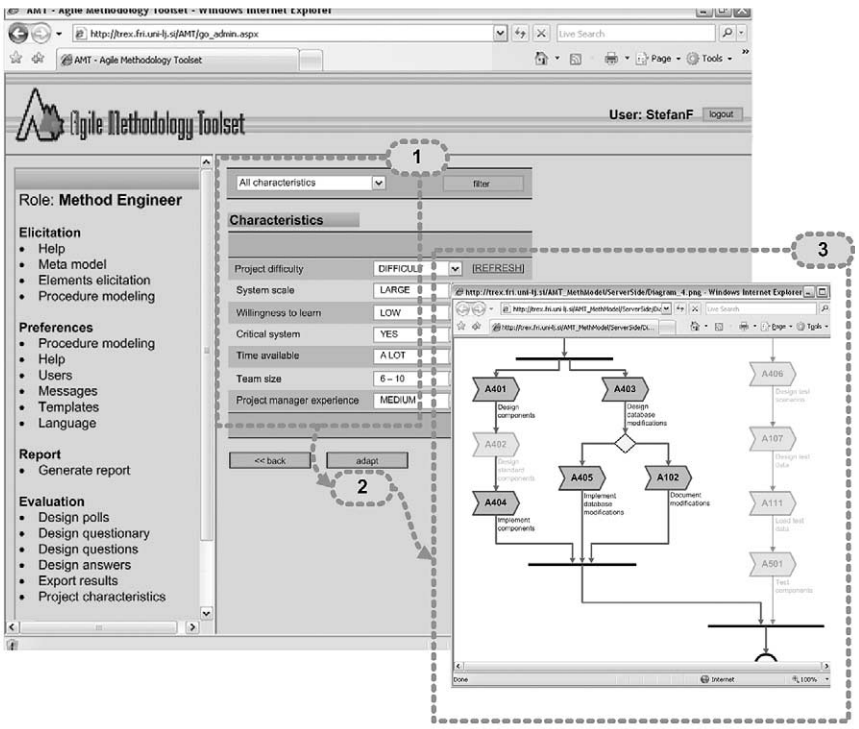
5.2. *MethAdapt*

MethAdapt facilitates method tailoring using the rules that were set during method procedure elicitation. Based on these rules and project specifics MethAdapt creates an instance of the method that is tailored to the project characteristics (see also Subsection 4.2).

Fig. 8 depicts MethAdapt user interface. At the start of a new project, a method engineer sets the project characteristics (1), which are used by the rule processor to determine the most suitable method configuration (2). Based on the results of the rule processor MethAdapt creates the tailored instance of the method. The result of the tailoring is presented on the activity diagram on which the activities that should not be executed are depicted in light colour (3).

5.3. *MethEval*

The main purpose of MethEval module is to facilitate continuous evaluation of the method. The evaluation is carried out in a series of surveys that are conducted among all users of the method. MethEval comprises two submodules. The first submodule allows method engineer to configure survey questionnaires and analyse the survey results. The second submodule generates survey questionnaires and distributes them among method

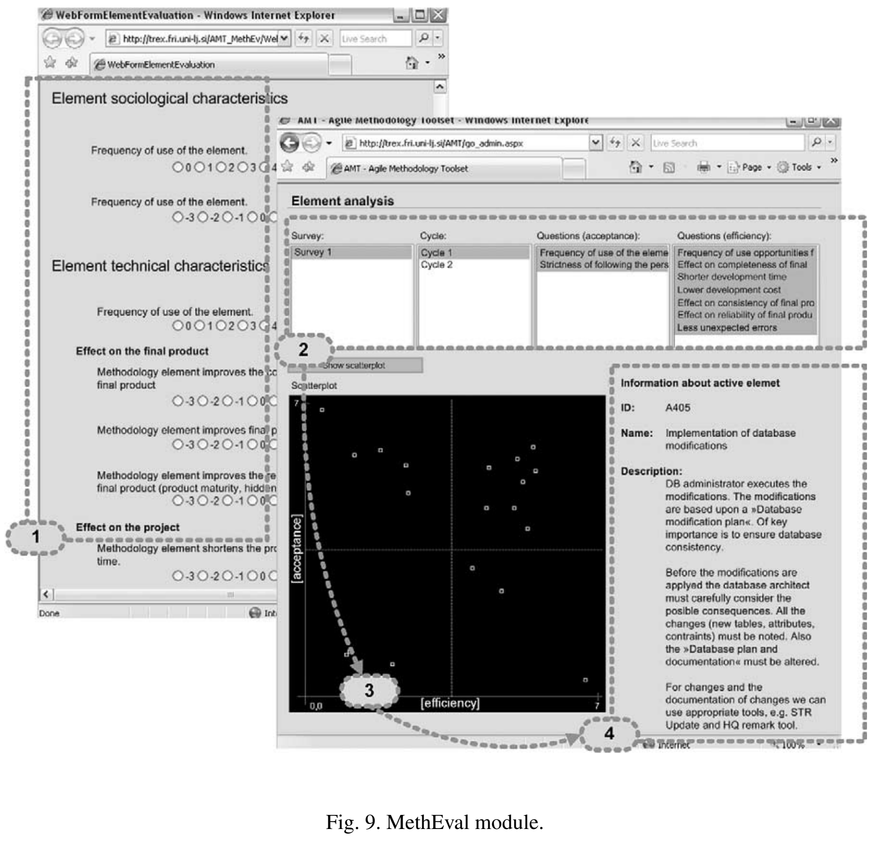


Project difficulty	DIFFICULT
System scale	LARGE
Willingness to learn	LOW
Critical system	YES
Time available	A LOT
Team size	6 – 10
Project manager experience	MEDIUM

Fig. 8. MethAdapt and a part of a tailored method.

users. Each method user is assigned only the questionnaires regarding those method elements that he uses in the context of his role. For instance, a programmer evaluates only programming activities, tools, etc., an analyst evaluates only analytical activities, tools, etc.

Left screenshot in Fig. 9 depicts a part of a generated survey questionnaire (1). After a method user completes the questionaire the results are stored in the common database. These results are later used in the analysis (right screenshot in Fig. 9). MethEval provides an interface for basic survey analysis. After a survey, a survey cycle and questions that should be analysed are selected (2), a scatter plot diagram is created that shows all evaluated element instances as dots (3). The diagram enables the method engineer to get a general impression of elements evaluation and to pinpoint the elements that are less suitable and require improvement. By clicking on an element dot the details of the element are displayed (4). For an advanced analysis MethEval provides an export function so that survey data can be exported in a format that is supported by well known statistical packages like SPSS.


WebFormElementEvaluation - Windows Internet Explorer
http://trex.fri.uni-lj.si/AMT_MethEv/Wel
Live Search
WebFormElementEvaluation
AMT - Agile Methodology Toolset - Windows Internet Explorer
http://trex.fri.uni-lj.si/AMT/go_admin.aspx
AMT - Agile Methodology Toolset
Page
Tools
Show scatterplot
Scatterplot
7
[acceptance]
0,0
[efficiency]
7
Done
Internet
100%
1
2
3
4

Element sociological characteristics

Frequency of use of the element.
○0 ○1 ○2 ○3 ○4

Frequency of use of the element.
○-3 ○-2 ○-1 ○0

Element technical characteristics

Frequency of use of the element.
○0 ○1 ○2 ○3 ○4

Effect on the final product

Methodology element improves the co
final product
○-3 ○-2 ○-1 ○0

Methodology element improves final p
○-3 ○-2 ○-1 ○0

Methodology element improves the re
final product (product maturity, hidden
○-3 ○-2 ○-1 ○0

Effect on the project

Methodology element shortens the pro
time.
○-3 ○-2 ○-1 ○0

Element analysis

Survey:	Cycle:	Questions (acceptance):	Questions (efficiency):
Survey 1	Cycle 1	Frequency of use of the eleme	Frequency of use opportunities f
	Cycle 2	Strictness of following the pers	Effect on completeness of final
			Shorter development time
			Lower development cost
			Effect on consistency of final pro
			Effect on reliability of final produ
			Less unexpected errors

Information about active elemet

ID: A405

Name: Implementation of database modifications

Description:
DB administrator executes the modifications. The modifications are based upon a »Database modification plan«. Of key importance is to ensure database consistency.

Before the modifications are applyed the database architect must carefully consider the posible consequences. All the changes (new tables, attributes, contraints) must be noted. Also the »Database plan and documentation« must be altered.

For changes and the documentation of changes we can use appropriate tools, e.g. STR Update and HQ remark tool.

Fig. 9. MethEval module.

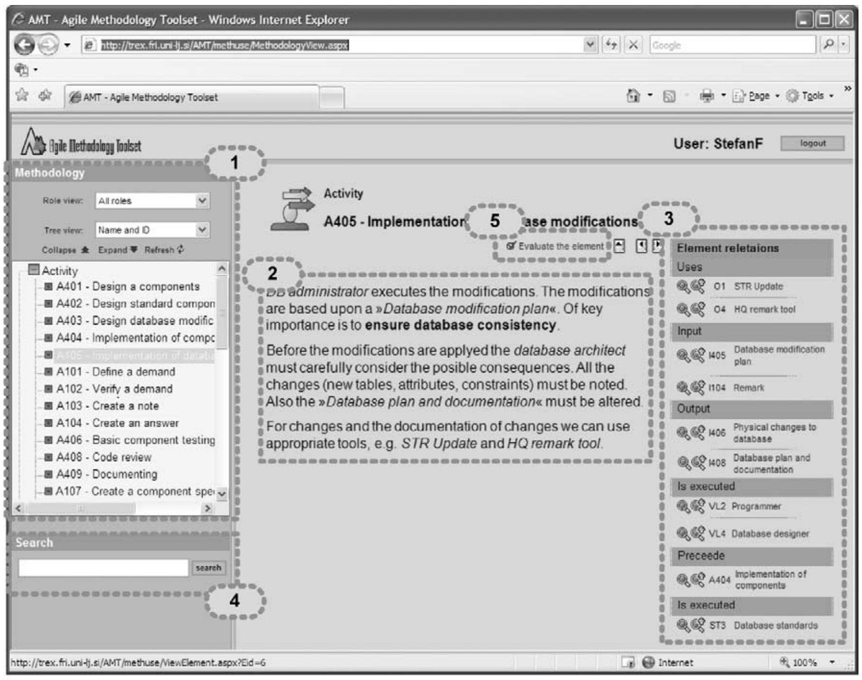
5.4. *MethGen in MethUse*

The purpose of MethGen and MethUse modules is to present the method to its end-users.

MethGen is a relatively simple module that uses the data stored in the common database to generate a method reference book in PDF format suitable for printing. Such reference book can either be used by the method users to learn the method or can be given to other parties interested, e.g., a customer.

MethUse, however, is a more complex module that enables dynamic access to the method content. Typically, MethUse is used by the method users who require access to the electronic method reference guide. The main purpose of the module is to make access to the method content as easy and quick as possible.

Fig. 10 shows the screenshot of MethUse module. The method content is accessed through navigation panel (1). The content of the panel can be filtered to show only the element instances that are in a relation to a certain role instance. After a user chooses a certain element instance, its description (2) and relations to other element instances (3) is displayed. The user can click on any of the relations (3) to see the description of the related element instance. This way the user can quickly find all the information about an element instance and its relations. Apart form using the navigation panel the user can also search through the method content using the search panel (4) which lists all the elements that match the search criteria.


AMT - Agile Methodology Toolset - Windows Internet Explorer
http://trex.fri.uni-lj.si/AMT/methuse/MethodologyView.aspx
AMT - Agile Methodology Toolset
Page
Tools
Agile Methodology Toolset
User: StefanF
logout
1
Methodology
Role view: All roles
Tree view: Name and ID
Collapse Expand Refresh
Activity
A401 - Design a components
A402 - Design standard compon
A403 - Design database modific
A404 - Implementation of compo
A405 - Implementation of databa
A101 - Define a demand
A102 - Verify a demand
A103 - Create a note
A104 - Create an answer
A406 - Basic component testing
A408 - Code review
A409 - Documenting
A107 - Create a component spe
Search
search
4
Activity
A405 - Implementation
5
ase modifications
3
Evaluate the element
2
Element reletaions
Uses
O1 STR Update
O4 HQ remark tool
Input
I405 Database modification plan
I104 Remark
Output
I406 Physical changes to database
I408 Database plan and documentation
Is executed
VL2 Programmer
VL4 Database designer
Preceede
A404 Implementation of components
Is executed
ST3 Database standards
http://trex.fri.uni-lj.si/AMT/methuse/ViewElement.aspx?Eid=6
Internet
100%

DB administrator executes the modifications. The modifications are based upon a »*Database modification plan*«. Of key importance is to **ensure database consistency**.

Before the modifications are applyed the *database architect* must carefully consider the posible consequences. All the changes (new tables, attributes, constraints) must be noted. Also the »*Database plan and documentation*« must be altered.

For changes and the documentation of changes we can use appropriate tools, e.g. *STR Update* and *HQ remark tool*.

Fig. 10. MethUse module.

An important part of MethUse is also its link to MethEval module. In case, that the role of the current MethUse user is related to the displayed element instance, the evaluation link is shown (5). This indicates that the user is entitled to evaluate the element instance. By clicking the evaluation link the user can open the survey questionnaire where he evaluates the element instance (see also Fig. 9).

6. Conclusions and Further Work

In this paper we presented a framework tool support for reengineering software development methods. Using the framework organisations can reengineer their existing ways of working and establish formalised methods that are *organisation-specific* and *auto-adjustable* to specifics of their projects.

In respect to the method engineering field the contribution of the framework should be seen in the integration of the method engineering principles within the software process improvement scenario. This way we assure the improved methods are not rigid but adjustable to specific circumstances. Furthermore, the framework encapsulates activities for continuous method evaluation and improvement based on the organisation's technical and social characteristics. Specifically the latter have been very often neglected by the traditional approaches to method engineering.

There are several directions in which we tend to continue the existing research work. Firstly, we wish to extend the framework to cover not only the creation and configuration of software development processes but rather arbitrary IT processes or even business processes. The research on this subject has started and is reported in a separate paper submitted to this conference. Next, we wish to improve the framework by incorporating a repository of best practices in software development which will facilitate (following assembly-based method engineering principles) semi-automatic creation of base methods. Finally, our goal is to employ the framework, specifically the method configuration phase, in the research project aimed at software development in rapidly created virtual teams.

References

Ajzen, I. (1991). The theory of planned behavior. *Organizational Behavior and Human Decision Processes*, **50**, 179–211.

Ayed, M.B., J. Ralyte and C. Rolland (2004). Constructing the Lyee method with a method engineering approach. *Knowledge-Based Systems*, **17**(7–8), 239–248.

Bajec, M., and M. Krisper (2005). A methodology and tool support for managing business rules in organisations, *Information Systems*, **30**(6), 423–443.

Bajec, M., D. Vavpotič and M. Krisper (2004). The scenario and tool-support for constructing flexible, people-focused systems development methodologies. In *Proc. ISD'04*, Vilnius, Lituania.

Bajec, M., D. Vavpotič and M. Krisper (2007). Practice-driven approach for creating project-specific software development methods. *Information and Software Technology*, **49**(4), 345–365.

Brinkkemper, S., K. Lyytinen and R.J. Welke (1996). Method engineering: principles of method construction and tool support. In S. Brinkkemper, K. Lyytinen and R.J. Welke (Eds.), *Conf. on Principles of Method Construction and Tool Support*. Selected papers. Kluwer Academic Publishers, Boston, MA.

Caplinskas, A., and J. Gasperovic (2005). Techniques to Aggregate the Characteristics of Internal Quality of an IS Specification Language, *Informatica*, **16**(4), 519–540

CMU/SEI-2002-TR-029 (2002). *Capability Maturity Model ®Integration (CMMISM)*, Version 1.1. SEI.

Fitzgerald, B. (1998). An empirical investigation into the adoption of systems development methods. *Information & Management*, **34**(6), 317–328.

Hardy, C.J., J.B. Thompson and H.M. Edwards (1995). The use, limitations and customization of structured systems development methods in the UK. *Information and Software Technology*, **37**(9), 467–477.

Harmsen, F., S. Brinkkemper and H. Oei (1994). Situational Method Engineering for IS project approaches. In A. Verrijn–Stuart and T.W. Olle (Eds.), *Methods and Associated Tools for the IS Life Cycle*. Elsevier, pp. 169–194.

Harter, D.E., M.S. Krishnan and S.A. Slaughter (2000). Effects of process maturity on quality, cycle time, and effort in software projects. *Management Science*, **46**(4), 451.

Herbsleb J.D., D.R. Goldenson (1996). A systematic survey of CMM experience and results. In *18th International Conference on Software Engineering*.

Huisman, M., and J. Iivari (2002). The individual deployment of systems development methods, *Lecture Notes in Computer Science*, **2348**, 134–150.

Huisman, M., J. and Iivari (2003). The organizational deployment of systems development methods. In *Information Systems Development: Advances in Methods, Components, and Management*. Kluwer, pp. 87–99.

ISO/IEC-15504 (1998). *Information Technology – Software Process Assessment*.

ISO/IEC-FCD-9126-1 (1998). *Software Product Quality*, Part 1, Quality model.

Kapocius, K., and R. Butleris (2005). Repository for business rules based IS requirements. *Informatica*, **17**(4), 503–518.

Karlsson, F., and P.J. Ågerfalk (2004). Method configuration: adapting to situational characteristics while creating reusable assets. *Information and Software Technology*, **46**(9), 619–633.

Kelly, S. (1997). *Towards a Comprehensive MetaCASE and CAME Environment – Conceptual, Architectural, Functional and Usability Advances in MetaEdit+*. University of Jyväskylä, Finland, Jyväskylä.

Mathiassen, L. (2002). Collaborative practice research. *Information Technology and People*, **15**, 321–345.

Middleton, P. (1999). Managing information system development in bureaucracies. *Information and Software Technology*, **41**(8), 473–482.

Niazi, M., D. Wilson and D. Zowghi (2005). A maturity model for the implementation of software process improvement: an empirical study. *Journal of Systems and Software*, **74**(2), 155–172.

Parzinger, M.J., and R. Nath (2000). A study of the relationships between total quality management implementation factors and software quality. *Total Quality Management*, **11**(3), 353–371.

Paulk, M.C., B. Curtis, M.B. Chrisis and C.V. Weber (1993). *Capability Maturity Model for Software*, version 1.1. CMU/SEI-93-TR-24, February, Software Engineering Institute.

Pressman, R.S. (2004). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, New York.

Ralyté, J., R. Deneckère and C. Rolland (2003). In J. Eder *et al.* (Eds.), *Towards a Generic Model for Situational Method Engineering* (CAiSE 2003), Klagenfurt, Austria, Springer, Haidelberg, pp. 95–110.

Rogers, E.M. (2003). *Diffusion of Innovations*, Free Press, New York.

Vavpotič, D., M. Bajec and M. Krisper (2004). Measuring and improving software development method value by considering technical and social suitability of its constituent elements. In O. Vasilecas and J. Zupančič (Eds.), *Advances in Theory, Practice and Education, Proc. of the 13th Inter. Conf. on IS Development*, Technika, Vilnius, pp. 228–238.

Vavpotič, D., M. Bajec and M. Krisper (2006). Scenarios for improvement of software development methodologies. In A.G. Nilsson, R. Gustas, W. Wojtkowski, W.G. Wojtkowski, S. Wrycza and J. Zupancic (Eds.), *Advances in Information Systems Development*, Vol. 1, Bridging the gap between academia and industry, Springer, New York, pp. 278–288.

Venkatesh, V., and F.D. Davis (2000). A theoretical extension of the Technology Acceptance Model: Four longitudinal field studies. *Management Science*, **46**(2), 186–204.

M. Bajec is an assistant professor at the University of Ljubljana, Faculty of Computer and Information Science. He received his MSc and PhD degrees from the Faculty of Computer and Information Science in 1998 and 2001 respectively. He delivers courses on information systems, information systems development, and databases. His main research interests include software development methodologies, IT/IS strategy planning and business process reengineering and renovation. His research work has been published in local and international journals. Marko Bajec is president-elect of the Slovenian Chapter of the *Association for Information Systems*, and vice-president of the *Slovenian Society Informatika*.

D. Vavpotič is a senior-lecturer of information systems in the Faculty of Computer and Information Science at the University of Ljubljana. He received his MSc and PhD degrees from the Faculty of Computer and Information Science in 2003 and 2006 respectively. He delivers courses on information systems and information systems development. His primary research interests include software development methods, approaches to method evaluation, and agile methods. His research has appeared in *Information and Software Technology*, *Applied Informatics*, and *Electro-Technical Review*.

Karkasas ir jį palaikantys instrumentai, skirti programinės įrangos kūrimo metodų reinžinerijai

Marko BAJEC, Damjan VAVPOTIČ

Šiame straipsnyje siūlomi karkasas ir jį palaikantys instrumentai, padedantys programinę įrangą kuriančioms bendrovėms įdiegti pas save jų poreikius tenkinančius techniškai ir socialiniu požiūriu korektiškus formalizuotus darbo metodus. Pasinaudodamos tuo karkasu, bendrovės gali įvertinti ir patobulinti esamus jų darbo metodus, formalizuoti tuos metodus ir, pasinaudodamos kaupiama savo darbo patirtimi, nuolat juos praturtinti. Dar daugiau, šitaip sukonstruoti formalizuoti darbo metodai yra lankstūs ir, pasinaudojant karkasą palaikančiais instrumentais, gali būti automatiškai pritaikomi specifiniams konkretaus projekto ar konkretaus kolektyvo poreikiams. Straipsnyje aprašyta siūlomo karkaso idėja ir to karkaso instrumentinis palaikymas.