Software Process Improvement Based on the Method Engineering Principles

Marko Bajec, Damjan Vavpotič, Štefan Furlan and Marjan Krisper
University of Ljubljana, Faculty of Computer & Information Science
Trzaska 25, 1000 Ljubljana, Slovenia
{marko.bajec, damjan.vavpotic, stefan.furlan, marjan.krisper}@fri.uni-lj.si

Abstract. While it used to be a common belief that the use of rigorous methods in software development is beneficial if not compulsory to assure success of software development projects, the investigations in practice reveal developers often avoid to follow prescribed methods and that there is a wide gap between the organisations' official methods and the work actually performed by their developers in IT projects. According to the literature, there are many reasons contributing to this rather undesirable situation. The two of them are *rigidity* of methods and their *social inappropriateness*. In the *MasterProc* project we have addressed these issues by developing a framework and tool-support for the reengineering of software development methods. Using the framework an organisation can reengineer its existing ways of working into a method that is *organisation-specific* and *auto-adjustable* to specifics of its projects. The evaluation that was performed in five partner companies is motivating, as it shows the framework can be very useful in improving software development practice. This paper describes the framework philosophy and its main components.

1 Introduction

It was decades ago when the software development became acknowledged as a complex process that needed disciplined methodological approaches. Since then a number of software development methods have emerged. Interestingly, in the last ten years, software development methods are not seen anymore as a panacea for software development and the wave of enthusiasm about their practical value has started to decrease. It has been empirically proved that in real practice the use of methods is actually low (see e.g. [1 – 5]). In the research community, several reasons have been identified as explanatory for this situation (see e.g. [4 – 5]). The two of them that seem to be the most important are: *inflexibility*, which is a characteristic of

Please use the following format when citing this chapter:

Bajec, M., Vavpotič, D., Furlan, Š., Krisper, M., 2007, in IFIP International Federation for Information Processing, Volume 244, Situational Method Engineering: Fundamentals and Experiences, eds. Ralyté, J., Brinkkemper, S., Henderson-Sellers B., (Boston Springer), pp. 283-297.

methods that permits virtually no adjustments to specific circumstances (a.k.a. *rigidity*), and *social inappropriateness*, i.e. unsuitability of the prescribed method to the company's actual performance or to the characteristics of the company's development team.

In this paper we present a framework for reengineering software development methods that we have developed under the *MasterProc* project[1]. Building on the established principles of the software process improvement initiatives and specifically of the method engineering, the framework facilitates companies that wish to improve their software development processes with guidelines and tools for acquiring their ways of working, for their continuous improvement, and for their adaptation to circumstances of a particular project or team.

The paper is organised as follows. In Section 2 we describe the research approach adopted in 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 in Sections 4 where the philosophy and main components of the suggested framework are described. 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* [6] 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.

[1] The MasterProc is a research project which is carried out under the umbrella of the Centre of excellence. The project was co-founded by the Slovenian Ministry of Higher Education, Science and Technology, European Commission and the participating Software Companies.)

3 Related work

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 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), which represents a central framework for software quality and process improvement (see e.g. [7-8]). 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 *optimised*. 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 [9-10], 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

[2] The description of CMM maturity levels is based on [7].

process that can lead to higher levels of CMM maturity, i.e. level 4 (managed) and level 5 (optimised).

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 [11] 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 [12]. Also, different repositories have been proposed for their storage (e.g. [11-13]). 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 [13], Ralyté describes 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 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 [14]. 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* [13] a.k.a. *evolution-based approach* [15]. 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 [16].

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. [4, 17]). 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 [17] and [16].

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 [18]. 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.

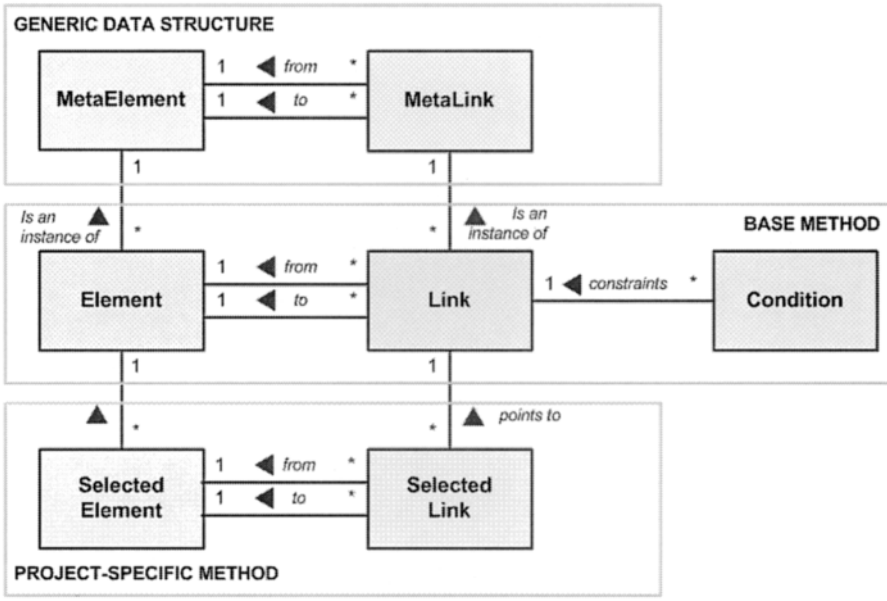

Fig. 1. A generic data structure

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

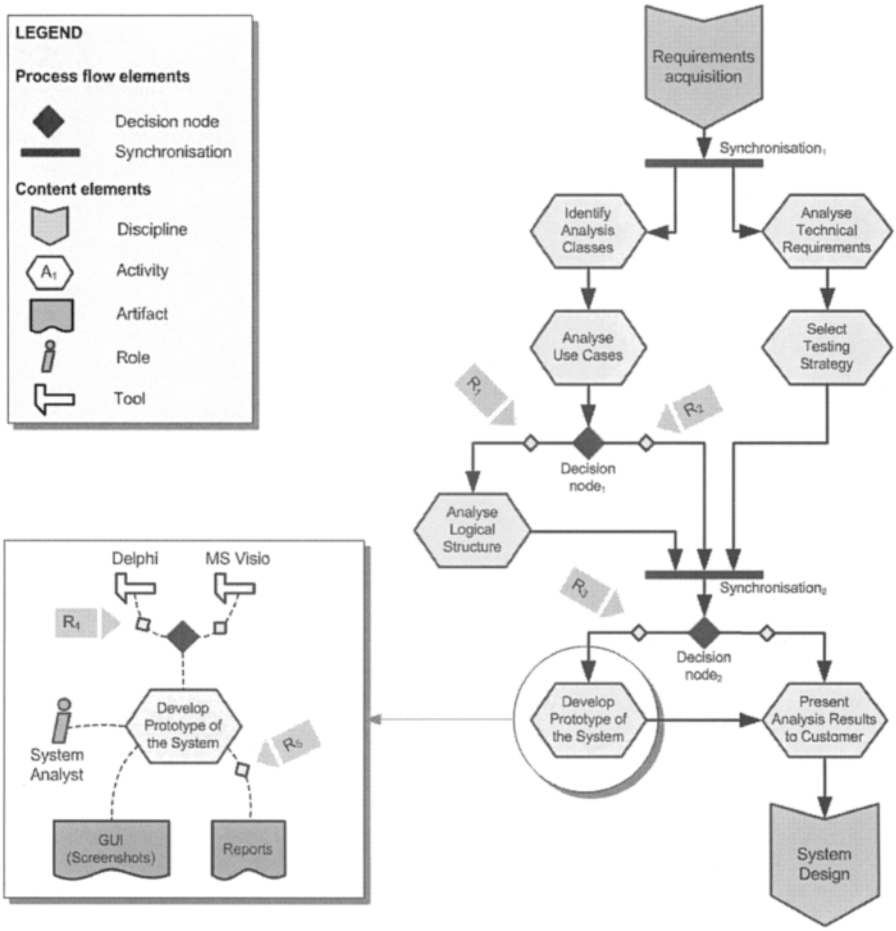

Fig. 2. Representation of a base method

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 [12]). 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*.
- 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

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

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.

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 [16].

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 [19 – 21] or only social [22 – 24] 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.

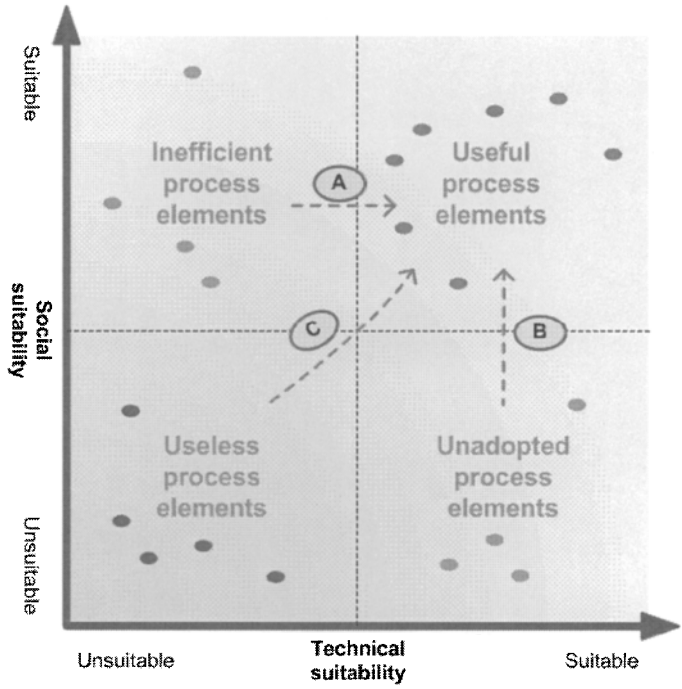

Fig. 3. Application of the evaluation model

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 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.
- 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 not suitable can be improved by using different *improvements scenarios.* These depend 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.

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

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 [18] and [25].

5 Conclusions and further work

In this paper we presented a framework 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.

6 References

1. C. J. Hardy, J. B. Thompson, and H. M. Edwards, The use, limitations and customization of structured systems development methods in the UK, Information and Software Technology, 37(9), 467-477 (1995).
2. M. Huisman and J. Iivari, The individual deployment of systems development methods, Lecture Notes in Computer Science, (Springer 2348, 134-150, 2002).
3. M. Huisman and J. Iivari, The organizational deployment of systems development methods, Information Systems Development: Advances in Methods, Components, and Management, (Kluwer 87-99, 2003).
4. B. Fitzgerald, An empirical investigation into the adoption of systems development methods, Information & Management, 34(6) 317-328 (1998).
5. P. Middleton, Managing information system development in bureaucracies, Information and Software Technology. 41(8), 473-482 (1999).
6. L. Mathiassen, Collaborative practice research. Information Technology and People, 15, 321-345 (2002).
7. M.C. Paulk, B. Curtis, M. B. Chrisis, C. V. Weber, Capability Maturity Model for Software, version 1.1, CMU/SEI-93-TR-24, February, Software Engineering Institute (1993).
8. R. S. Pressman, Software Engineering: A Practitioner's Approach. McGraw-Hill, New York (2004).
9. D. E. Harter, M. S. Krishnan, S. A. Slaughter, Effects of process maturity on quality, cycle time, and effort in software projects. Management Science April 46 (4), 451 (2000).

10.M. J. Parzinger, R. Nath, R., A study of the relationships between total quality management implementation factors and software quality. *Total Quality Management* 11 (3), 353–371 (2000).
11. F. Harmsen, S. Brinkkemper, and H. Oei, Situational Method Engineering for IS Project Approaches, in: Methods and Associated Tools for the IS Life Cycle, edited by A. Verrijn-Stuart and T. W. Olle (Elsevier, 1994), pp. 169 – 194.
12. S. Brinkkemper, K. Lyytinen, and R. J. Welke, Method engineering: principles of method construction and tool support. Conf. on Principles of Method Construction and Tool Support, selected papers. Edited by S. Brinkkemper, K. Lyytinen, and R. J. Welke, (Kluwer Academic Publishers, Boston, MA, 1996).
13. J. Ralyté, R. Deneckère, and C. Rolland, Edited by J. Eder et al, Towards a generic model for situational method engineering (CAiSE 2003), Klagenfurt, Austria, June 16-18, 2003, (Springer, Haidelberg, 2003), pp 95-110.
14. F. Karlsson, and P. J. Ågerfalk, Method configuration: adapting to situational characteristics while creating reusable assets, *Information and Software Technology*, 46(9), 619-633 (2004).
15. M. B. Ayed, J. Ralyte, C. Rolland, Constructing the Lyee method with a method engineering approach, *Knowledge-Based Systems* 17(7-8), 239-248 (2004).
16. M. Bajec, D. Vavpotič, M. Krisper, Practice-driven approach for creating project-specific software development methods, *Information and Software Technology*, 49(4), 345-365 (2007).
17. M. Bajec, D. Vavpotič, and M. Krisper, The scenario and tool-support for constructing flexible, people-focused systems development methodologies, in: Proc. ISD'04, Vilnius, Lituania, (2004).
18. D. Vavpotič, M. Bajec, M. Krisper, Measuring and improving software development method value by considering technical and social suitability of its constituent elements, in: Advances in theory, practice and education: proc. of the 13th Inter. Conf. on IS Development, edited by O. Vasilecas, J. Zupančič, (Technika, Vilnius, 2004), pp. 228-238.
19. CMU/SEI-2002-TR-029, Capability Maturity Model ® Integration (CMMISM), Version 1.1. SEI., (2002)
20. ISO/IEC-15504, Information technology – software process assessment, (1998)
21. ISO/IEC-FCD-9126-1, Software product quality – Part 1: Quality model, (1998)
22. E. M. Rogers, Diffusion of innovations, (Free Press, New York, 2003).
23. I. Ajzen, The Theory of Planned Behavior, *Organizational Behavior and Human Decision Processes*, 50, 179-211 (1991).
V. Venkatesh, and F. D. Davis, A theoretical extension of the Technology Acceptance Model: Four longitudinal field studies, *Management Science* 46(2), 186-204 (2000).
24. D. Vavpotič, M. Bajec, M. Krisper, Scenarios for improvement of software development methodologies, in: Advances in information systems development. Vol. 1, Bridging the gap between academia and industry, edited by A.G. Nilsson, R. Gustas, W. Wojtkowski, W.G. Wojtkowski, S. Wrycza and J. Zupancic, (Springer, New York, 2006), pp. 278-288.