

Situational Evaluation of Method Fragments: an Evidence-Based Goal-Oriented Approach

Hesam Chiniforooshan Esfahani¹, Eric Yu², Jordi Cabot³

¹Department of Computer Science, University of Toronto

²Faculty of Information, University of Toronto, ³INRIA - École des Mines de Nantes

¹hesam@cs.toronto.edu, ²yu@ischool.utoronto.ca, ³jordi.cabot@inria.fr

Abstract. Despite advances in situational method engineering, many software organizations continue to adopt an ad-hoc mix of method fragments from well-known development methods such as Scrum or XP, based on their perceived suitability to project or organizational needs. With the increasing availability of empirical evidence on the success or failure of various software development methods and practices under different situational conditions, it now becomes feasible to make this evidence base systematically accessible to practitioners so that they can make informed decisions when creating situational methods for their organizations. In this sense, this paper proposes a framework for evaluating the suitability of candidate method fragments prior to their adoption in software projects. The framework makes use of collected knowledge about how each method fragment can contribute to various project objectives, and what requisite conditions must be met for the fragment to be applicable. Pre-constructed goal models for the selected fragments are retrieved from a repository, merged, customized with situational factors, and then evaluated using a qualitative evaluation procedure adapted from goal-oriented requirements engineering.

Keywords. Software Development Methodology, Situational Method Engineering, Goal-Oriented Modeling, Method Evaluation

1 Introduction

One of the common concerns of project managers is to ensure that their development processes fit well with project situations. Despite the proposal of elaborate frameworks for building situational methods [1-4], many software companies still follow ad-hoc methods of software development, which are built intuitively by adopting some fragments from different methodologies and tailoring them to their development method [5, 6]. With this approach, the best case scenario is that the company will benefit from all advocated advantages of selected method fragments. Unfortunately, this is not always the case, and there are numerous reports of project failure that were due to the improper choice of development method [7].

For instance, suppose that a software organization wants to incorporate “Pair Programming” [8] as part of its development method. It is known that pair programming usually helps to achieve some objectives such as “reduced defect rate” and “real-time knowledge transfer”. But, are these objectives fully achievable in all project situations? Could other method fragments, when combined with pair programming, facilitate (or hamper) these objectives? There are increasing empirical studies that have addressed such questions, and investigated the success or the failure of different method fragments in different project situations. For instance, there exist over 100 empirical studies about “Pair Programming” in various situations¹. Considering the large number of existing method fragments and their volume of supporting studies, the need arises for a solution that can systematically make this evidence-based information available to practitioners, and help them during the process of method construction.

This paper proposes a framework for evaluating software development processes. The proposed framework considers a development method as a set of method fragments [2], and helps project managers anticipate whether the expected benefits in adopting the selected fragments will be attainable given the particular circumstance faced by the project. Starting from a set of candidate fragments, the framework leads the user to consider their objectives and requisites. Then, method objectives will be evaluated by reflecting the impacts of situational factors on method requisites, and propagating the achievement status of method requisites to method objectives.

In order to take advantage of existing situational knowledge of different method fragment, the proposed framework provides *evidence-based* repositories, which contain a structured representation of results from empirical studies. The contents of these repositories are gathered through systematic literature review [9]. The evidence base of this framework visualizes the collected knowledge of method fragments in a *goal-oriented* representation. The framework reuses its ready-made goal models to facilitate the process of method evaluation. The proposed framework can be deployed as a complement to existing method engineering frameworks, and due to its simplicity, it can also be used by project managers, who are not willing to involve into the detailed steps of situational method construction. In the rest of this paper, we will explain the framework using an illustrative scenario, introduced in the next section.

2 Motivating Scenario

As an illustrative scenario, consider a project manager facing the following circumstances in her project: (1) Technology used in the project is new to the company; (2) Developers are mainly junior programmers; (3) Team members are distributed, with a project wiki and email-lists being the main communication channels; (4) the project manager serves as the customer representative; and (5) Developers do not have access to the target device on which the product software will be installed, and can only test the software on a limited emulator.

¹ The search was run on libraries of IEEE Explorer, Springer, and Elsevier in November 2009

The project manager is attracted to agile methods as they are said to lead to faster time to market, improved communication, more reliable project planning, and more effective teamwork (e.g. efficient self-organizing). Based on what she knows about agile methods and the characteristics of her organization, the manager is planning to adopt the following method fragments from XP [8] and Scrum [10]: (1) Daily Scrum Meeting: Short daily meetings to explain performed activities and to discuss obstacles; (2) Short Iterations: Delivering an increment of software every three weeks; (3) Iteration Planning: Selecting high priority requirements at the beginning of each iteration, and breaking them down to smaller tasks; (4) Pair Programming: Assigning two programmers to work together on each software component. However, she is unsettled by many questions, such as:

- Will the selected method fragments work well together or will they conflict? (e.g. should pair programming be used together with iteration planning?)
- Does my team and project environment satisfy the necessary requisites to take advantage of the selected method fragments?
- Will the combined set of fragments produce the desired results and meet project objectives?

3 Evaluation Framework

This section introduces a framework for evaluating a set of candidate method fragments to be a part of organization development method. The framework helps project managers to anticipate the achievement of method objectives, and find answer for questions, such as those mentioned in previous section. Fig. 1 shows the overall structure of the framework. The existing knowledge about method fragments will be kept in an *Evidence Base*, which is composed of two repositories: (1) *Method Fragment Repository* that holds objectives and requisites of method fragments, along with situational evidences; and (2) *Model Fragment Repository* that holds the graphical representations of method fragments.

To evaluate a number of candidate method fragments, their corresponding model fragments will be retrieved from the Model Fragment Repository, and will be merged based on their objectives. Then the integrated models will be customized and initialized with respect to the project situational attributes. During this step, the situational evidences that have been stored in Method Fragment Repository will be reused. At the end, the integrated models will be evaluated, to anticipate the satisfaction degree of method fragments' objectives. The framework makes its evidence base available to software organizations, and process designers are responsible for performing the evaluation steps.

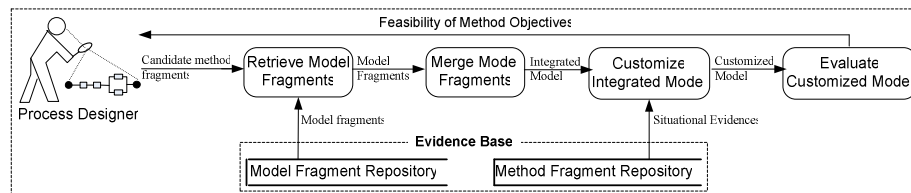


Fig. 1: Evaluation Framework

3.1 Evidence Base

The evidence base of this framework is inspired by the assembly-based SME frameworks [2, 11]. But, unlike current method repositories (e.g. OPF [12]) that store the specification of method fragments, the evidence base of this framework contains the experience results of empirical studies that have been conducted for different method fragments. This framework follows the definition of *Method Fragment* as explained in [2], which classifies method fragments into *process* and *product* fragments, respectively referring to the elements of development process and structure of products. Another category of method fragments has been introduced in [12], called *producer*, which represent methodology roles played by persons or tools. As mentioned before, the evidence base is composed of two repositories that will be explained in the next two subsections.

3.1.1 Method Fragment Repository

The Method Fragment Repository stores the textual information of method fragments. The information has been collected through systematic literature review of published empirical studies, which have tested the enactment of method fragments in different situations. For each method fragment, this repository holds two datasets: *Objectives Dataset*, and *Requisites Dataset*.

3.1.1.1 Objectives Dataset

For each method fragment, the repository provides an *objectives dataset*, which represents the quality goals that are expected to be achieved by the enactment of the method fragment. These objectives have been extracted from published empirical studies on method fragments (i.e., the dataset does not include quality goals that were just claimed for a method fragment without any supporting empirical evidence). This dataset stores method objectives in two categories: *major* and *minor*. A major objective is defined as a quality goal that can be decomposed into a number of sub-goals, called minor objectives. Perhaps the classification of quality goals could be performed more elaborately; however, for the sake of simplicity this framework considers only these two levels. The objectives dataset also provides situational evidences for the contribution of method fragment to its objectives. Besides, for every contribution relation, the dataset provides the reference to the study that provided the empirical evidence, and possibly the description of the situation of study. Table 1 shows a portion of the objective dataset for method fragment “Daily Scrum Meeting”.

Typically the contribution of a method fragment to its objectives is positive. However, there might be some situations where a method fragment adversely impacts its objectives. For instance, although “Daily Scrum Meetings” usually makes strong positive contribution to the “Improved Awareness” of a team about the activities of other team members, as studied in [S1], in the case of large projects with multiple development teams, daily meetings can cause confusion by bringing up excessive details, which are not relevant for a large portion of developers. The framework proposes four possible types of contribution relations: *strongly positive* (++), *positive* (+), *negative* (-), and *strongly negative* (--).

Table 1. A subset of major and minor objectives that “Daily Scrum Meeting” contributes to them, with reference to the investigating empirical studies, and particular situational evidences

Major Objective	Minor Objective	Contribution Type from Fragment	Study	Situation
Effective Communication		++	[S1]	Default
	Improved awareness (of what others are doing)	++	[S1]	Default
		-	[S1]	Large projects, as they may need extensive number of meetings
	Real-time knowledge transfer	+	[S8]	Default
		-	[S2, S12]	Distributed Development: use of email and wiki pages for comm.
	Enhanced Communication with business people / project leader	++	[S3, S8]	Existence of multi-level Scrum in case of many scrum teams
Better understanding of customer needs	+	[S8]	Default	

3.1.1.2 Requisites Dataset

The other dataset kept for each method fragment is the *requisites dataset*. This dataset contains the conditions that should be met for the successful enactment of a method fragment (e.g., resources to be provided, tasks to be performed, or personnel qualifications to be met). The framework defines the relation of a method fragment to its requisites as *decomposition relation*, since the successful enactment of a method fragment is due to the successful achievement of its requisites. Similar to the objectives dataset, this dataset represents method requisites in two levels of abstraction (major and minor), also sets the contribution relation of minor requisites to major ones. Besides the dataset provides situational evidences for each requisite, by referencing to the studies in which the requisite was satisfied or denied (partially or fully). In most cases it explains the significant situational factors of the referenced empirical studies that affected the fulfillment of method requisites. Fig. 2 (a) shows the metamodel of the method fragment repository.

Table 2 shows a subset of the requisites of “Pair Programming”, focused on “Effective Collaboration”. For instance, it shows that “Equal engagement (of pairs) in Coding” is a minor requisite that contributes positively to the major requisite “Effective Collaboration”. However, not every situation can satisfy this requisite. For example, the empirical study [S15] has shown that pairing programmers with different levels of expertise can result in passiveness of the weaker programmer, thus partial denial of the requisite. The objectives dataset also takes a goal-oriented approach in representing method requisites in order to facilitate their modeling and evaluation in later stages of the framework.

Table 2. A subset of requisites of “Pair Programming“, Contribution of Minor to Major Requisite; Situational Fulfillment Status [Satisfied(✓), Denied(×), or Partially Denied(⊗)]

Major Req.	Minor Requisite	Contrib. to Major Req.	Situa. Fulfill. Status	Study	Situation
Effective Collaboration	Equal engagement in coding	+	✓	[S15]	Pairing programmers with equal expertise
		+	⊗	[S15]	Pairing programmers with different expertise (weaker programmer became passive)
	Joint Decision Making	+	⊗	[S15]	Similar pairs; the one who had the control of machine usually had a significant advantage w.r.t decision making
	Collaboration be viable	+	✓	[S17]	Pairs with heterogeneous personality profile
			⊗	[S17]	Pairs with homogenous personality profile
Similar working and resting hours	+	×	[S30]	Pairs with different times for starting their job or resting	

3.1.2 Model Fragment Repository

The Model Fragment Repository of this framework contains the pre-constructed graphical representations of method fragments. For each method fragment, this repository contains a number of models, called *model fragments*, each visualizing a subset of its objectives and requisites (which were textually stored in the method fragment repository). In fact, each model fragment is a piece of a comprehensive model that could represent the whole knowledge of a method fragment. The main reason for breaking down the visualization of each method fragment was to avoid the potential complexities that could have emerged if the whole knowledge of a method fragment was represented in a single model.

The suitable modeling language for representing model fragments should have enough constructs to model method objectives and requisites, and also contribution and decomposition relations. We found the *i** modeling language [13] as an appropriate choice, especially by considering its prior use for modeling the intentional aspects of software processes [14]. For developing model fragments, the following set of *i** constructs are used: *Task*, representing method fragments; *Softgoal*, representing method objectives and requisites; *Contribution Link*, representing contribution relations; and *Decomposition Link*, representing decomposition relations. We defer the representation of actors to future work.

Fig. 2 (b) shows an example model fragment, developed for the method fragment: “Iteration Planning” and the major objective: “Effective Project Planning”. This model fragment shows that conducting iteration planning sessions (as described in Scrum [10]) would help to enhance the Effective project planning by positively contributing to the (1) realistic prioritization of tasks; (2) clear definition of iteration goals; and (3) accurate project scheduling. The meaning of contribution links used in model fragments corresponds to their counterparts in method fragment repository: Some+ and Help contribution links represent strong (++) and normal (+) positive

contributions; Some- and Hurt contribution links represent strong (--) and normal (-) negative contributions.

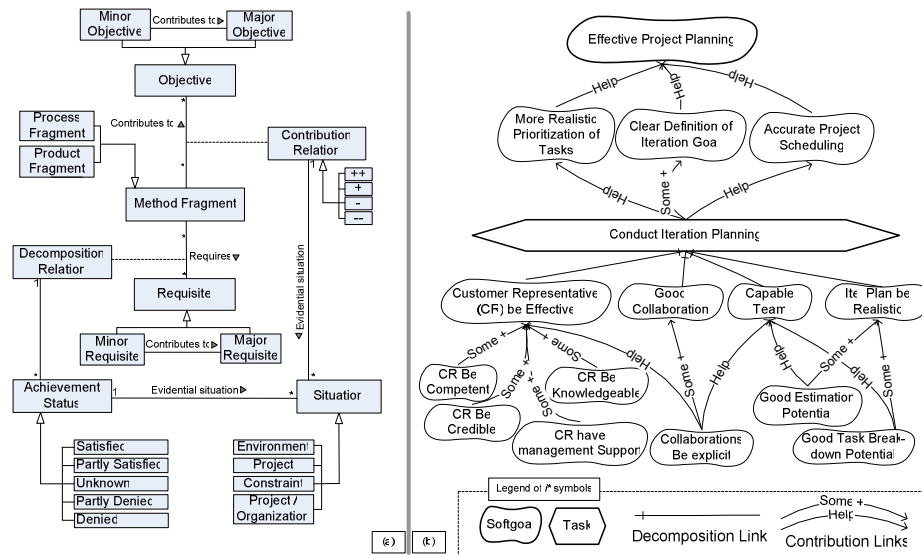


Fig. 2: (a) Metamodel of method fragment repository; (b) An instance of model fragment repository, representing the method fragment: “Iteration Planning” and the objective: “Effective Project Planning”

Since a model fragment usually represents only one of the major objectives of a method fragment, it contains a limited set of fragments’ requisites – those that are somehow related to the major objective. For instance, in Fig. 2 (b) if the objective of model fragment was “Effective Communication” (rather than “Effective Project Planning”), then requisites that were related to the capability of team members in task estimation and task breakdown, should have been discarded.

3.2 Situational Evaluation

So far we have described the evidence base of the framework. In the following subsections we describe the steps to be taken to evaluate a set of method fragments being considered for a given organization/project.

3.2.1 Model Merging

This framework evaluates a set of candidate method fragments by anticipating the extent to which their objectives will be satisfied. Some objectives of a method fragment may be in common with those of other method fragments. Thus, for correct anticipation of method objectives we have to consider the contribution of all candidate method fragments to those objectives. Model merging is intended to develop integrated views of model fragments, which highlight their contribution to the common set of objectives. In order to develop integrated models, process

designers should first specify a set of major objectives that they want to be evaluated. Then, retrieve the relevant model fragments from the Model Fragment Repository, and for every intended major objective, merge the related model fragments by following these steps:

- 1) Identify its minor objectives that are contributed to by candidate method fragments
- 2) Set the contribution relations from minor objectives to the major ones
- 3) Set contributions of method fragments to the minor objectives

While merging method fragments based on their objectives, we should also consider the potential contributions that may exist between the requisites of method fragments. Such contribution relations (if detected) should be represented in integrated models. The contribution relation of minor objectives to major ones (with respect to the method fragment) is proposed in the model fragment repository.

In our motivating scenario, project manager was interested to anticipate the satisfaction (or denial) degree of three major objectives: “Effective Communication”, “Effective Project Planning”, and “Efficient Self-Organizing”. Fig. 3 shows the result of merging three of our model fragments based on the common objective: “Effective Communication”.

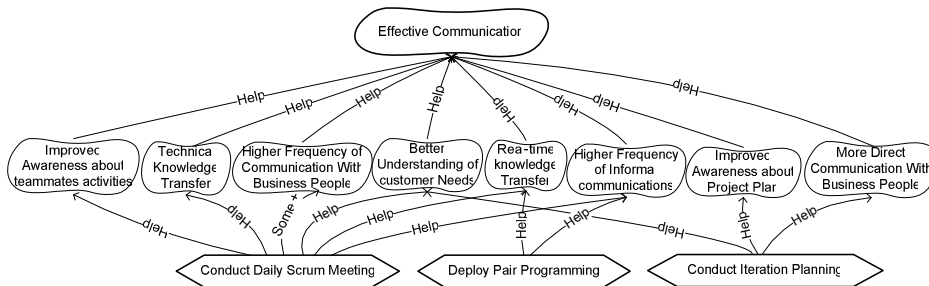


Fig. 3: Merging model fragments based on the common objective: “Effective Communication” (Requisites are not shown in this model)

Integrated models serve as intermediate artifacts in this framework. They can help process designers identify (1) method objectives that receive contradictory contributions via different method fragments; (2) method objectives that are strongly contributed, typically by different method fragments; and (3) method objectives that are weakly contributed, typically by only one method fragment. For instance, Fig. 3 shows that “Better Understanding of Customer Needs” is a method objective that receives contributions from two fragments “Daily Meeting” and “Iteration Planning”, thus we would expect its successful achievement according to the generic knowledge from the evidence base. However, will the specific situation of this project affect this assessment? To answer this question we need to first customize the integrated model according to the project situational factors and then complete its evaluation.

3.2.2 Model Customization

This phase is intended to customize the integrated goal models, in order to reflect the impacts of situational factors on them. The customization process takes place in two complementary stages: *Model Refinement*, which modifies the structure of the goal model; and *model initialization*, which assigns initial values to the requisites of method fragments. This framework proposes two strategies for model refinement:

- *Changing the type of contribution links* – regarding the specific situation of a software project, the types of contributions that a method fragment makes to its objectives might not remain as their default types (stored in the model fragments repository). In such cases, the contribution type should be altered to represent the specific situational attributes of software project. For instance, the default contribution of method fragment “Conduct Daily Meetings” (as defined in Scrum [10]) to its objective “Improved Awareness (of what others are doing)” is “Help”. However, in an organization where the size or the number of development teams is too large, this contribution should be altered to “Hurt” (explained in the reference [S1] of the Appendix).
- *Adding new softgoals to the requisites* – situational factors that impact the fulfillment of method requisites will be represented as new softgoals, contributing to the existing method requisites. Different categories of situational factors have been proposed in [2, 15, 16]. These can be represented as softgoals. For instance, “Established Standards” is a situational factor [2] that can be represented as softgoal “Standard X be followed”.

Different project situations imply different initial values for the achievement status of method requisites. This framework identifies two types of initialization for method requisites:

- *Evidential* – initializations that are supported by solid evidence(s). For instance, in distributed projects where development teams have substantial time zone differences, the achievement status of method requisite “Meetings be face-to-face” will be initialized with “Denied” value.
- *Assumptive* – Initializations that are based on assumptions with no significant evidence. For example, in situation that most of team members have just met each other, the achievement status of method requisite “Collaboration be Viable” cannot be predicted for sure, yet it can be assumed to become at least “Partially Satisfied”.

The empirical evidences, which have been populated in the repository of method fragments, can be quite helpful during model refinement and initialization, provided that a similar situation has been expressed.

Table 3 describes a number of significant situational factors of the illustrating scenario, their category (based on [2]), and their impact on model refinement. For initialization of model, both evidential and assumptive initializations have been used. For instance, since developers are junior programmers, we have enough evidences to initialize the “Experienced Programmers” to “Denied”. For requisites that we do not have solid evidences, we use assumptive initializations. For example, we assume the initial values of “Joint Decision Making” and “Collaboration be Viable” to be “Satisfied”. Fig. 4 shows the evidential initializations with green labels, and assumptive ones with red labels.

Table 3. Significant situational factors, their category based on [2] (Project/Organization, Project, Constraint, Environment), and their impact on customizing the goal model

Situational Factor	Category	Model Customization
Developers have little experience of software development	Project / Org.	Initialize the softgoal “Experienced Programmers” to “Denied”
The technology is new to the company	Project / Org.	Initialize softgoal “High Tech Knowledge” to “Partially Denied”
Developers are not always working at the same time/place	Project / Org.	Initialize the softgoal “Team members be Co-located” to “Partially Denied”
Face-to-face meeting with Scrum Masters are accessible only at certain times	Constraint	Add softgoal “Scrum Master be Available”, initialize to “Partially Denied”
Wiki pages are used for implementing daily meetings	Project	Add softgoal “Wiki Pages be Deployed” / Change the value of contribution links that connects “Daily Meeting” to “Higher Frequency of Com. With Business People”, and “Real-Time Knowledge Transfer” to “Hurt”
Use of emulator instead of actual devices	Environment	Add softgoal “Resources be Adequate” with initial value of “Partially Denied”

3.2.3 Model Evaluation

This phase iteratively propagates the initial (achievement) value of method requisites into the higher level elements of the integrated model. The proposed value propagation algorithm mainly follows the i^* forward evaluation algorithm [17] [18]. Value propagation will be performed with respect to the current value of contributing element and the value of contribution link, based on the propagation rules defined in Table 4. Since a contributed element might receive various values in this procedure, its “value bag” will be resolved based on the following value resolution rules [17]:

- Use the single label if the value bag has only one label
- Use the full label if:
 - The value bag has multiple labels of the same value
 - The value bag has labels with the same polarity with at least one full label
 - The previous human judgment produced a full label and new contribution is of the same value
- Use the minimum label if the bag has been filled through decomposition links. (Satisfied > Partially Satisfied > Conflict > Unknown > Partially Denied > Denied)
- Otherwise, use human judgment

Giorgini et al. in [19] introduced the concepts of *symmetric* and *asymmetric* contributions. When an element of a goal model symmetrically contributes to a softgoal, both its satisfaction and denial will be propagated to the softgoal. But in asymmetric contribution, the propagation will be performed either when the

contributor is satisfied, or denied. The i^* forward evaluation algorithm considers all of the contribution relations as symmetric. However, this may not be appropriate for negative contributions (Hurt, Some-). For instance, when a method fragment (e.g. “Pair Programming”) makes a Hurt contribution to a softgoal (e.g. “More LOC per programmer per hour”), the denial of method fragment would not necessarily result in partial satisfaction of softgoal, while its satisfaction would result in partial denial of softgoal. Here, we modified the i^* forward evaluation algorithm by considering positive contributions (Help, Some+) as symmetric, and negative contributions (Hurt, Some-) as asymmetric, applicable only when the contributor is (Partially) satisfied.

Table 4. Propagation Rules for Contribution Links (adopted from [17])

Original Label		Contribution Link Type				
Label	Name	Help	Hurt	Some +	Some -	Unknown
✓	Satisfiec	✓	✗	✓	✗	?
✓	Partially Satisfiec	✓	✗	✓	✗	?
?	Unknowr	?	?	?	?	?
✗	Partially Denied	✗	?	✗	?	?
✗	Deniec	✗	?	✗	?	?

Scenario: As an example of automatic resolution (not requiring human judgment), consider the objective “Better Understanding of Customer Needs”. The method fragment “Conduct Daily Meeting” has a “Help” contribution to this softgoal, and had been evaluated to “Partially Satisfied”. Therefore, based on the propagation rules of Table 4, a “Partially Satisfied” label will be added to the value bag of this softgoal. A similar label will be added to this bag, through the contribution of method fragment “Conduct Iteration Planning”. Therefore, this softgoal will be automatically evaluated to “Partially Satisfied”.

Manual resolution is needed where the automatic approach cannot be applied. In such cases, human judgment determines the value of the element, considering its value bag and status of its contributing elements. For example, consider the requisite softgoal “Be Productive” (a requisite of “Pair Programming”). This softgoal had received two “Partially Denied” and one “Partially Satisfied” labels through its contributing elements. We evaluated this softgoal to “Partially Denied”, based on the perception that the contributions of adequate resources and programming experience to the productivity of programmers are more significant than the similarity of their personality. Fig. 4 shows the result of evaluating an integrated model, developed for the “Effective Communication”.

4 Validity of Framework

Although the motivating scenario introduced in Section 2 was fictitious, we had experienced a similar scenario in a classroom setting. A software development method was created by combining the above-mentioned method fragments, and used for the major project in a second-year undergraduate computer science course, with 40 4-person teams. The project was about building an application for Blackberry devices. Most team members had little programming experience. Furthermore, the Scrum masters, who were course teaching assistants (TAs), had limited availability (their office hours), and were not so familiar with mobile programming.

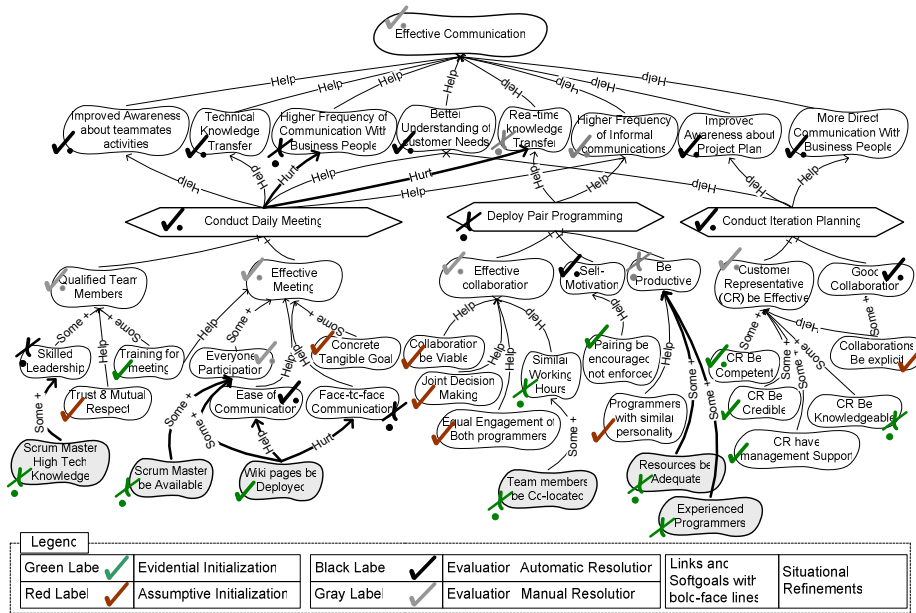


Fig. 4: Modification, Initialization, and Evaluation of an integrated model

As an initial test of framework validity, we compared the evaluation results generated by framework with what was observed in the classroom. For this comparison, we considered 18 minor objectives that were contributing to the three major objectives. The comparison showed that for 12 softgoals (66%), the framework-generated values were the same as the observed ones. For 4 of the softgoals (22%), the observed status of method objectives were better than what the framework suggested. In most cases the framework-generated values were “Partially Denied”, whereas the observed values were “Partially Satisfied”. For 2 of the softgoals (12%), the observed values were worse than what the framework anticipated. Thus overall, the framework produced results that were more conservative than actually observed.

5 Discussion and Future Work

This paper proposed a framework for anticipating the attainability of the objectives of a development method in a particular project situation. The framework aims to help project managers in deciding on a set of suitable method fragments. Methods for selecting method fragments have been proposed, e.g. Situation-Scenario-Success (S^3) [2] and Assembly Process Model (APM) [3], but they do not focused on the evaluation of objectives. A method fragment evaluation technique was proposed in [20], which also deals with capturing the evaluation experiences in method fragments repository. The proposed framework adopts a goal-oriented approach for analysis, modeling, and evaluation of method fragments. Similar approaches have been acknowledged in [19, 21, 22].

The reliance of this framework on the results of empirical studies has both positive and negative sides. On the positive side, it can help project managers by providing an evidence base, supported by the experiences of many practitioners and researchers on deploying different method fragments. On the negative side, reliance on the extensive body of empirical evidence can be subject to misinterpretation, inadequate or unreliable evidential data, and conflicting scenarios (e.g., same situation, different results). These negative effects can be moderated by relying mostly on empirical studies that had been peer reviewed and published in highly regarded conferences or journals and, if possible, whose results have been replicated.

One of the important factors that we considered in the design of this framework is that it should not impose excessive overhead to software organization. The provision of a model fragment repository is an initiative towards this end. Although this claim needs to be verified in our further industrial case studies, we expect that the pre-constructed model fragments facilitate the use of framework. So far, we have populated the framework repositories with an initial collection of 15 method fragments. More are being added in an ongoing project. As future work, we are developing automated tools to support the process of model merging, and will take advantage of the OpenOME² tool for the evaluation of goal models.

An underlying premise for this framework is to focus on method fragments rather than on the prescription of an entire process. This approach is in line with the idea of agility while doing software process improvement (SPI) [23]. Unlike traditional SPI frameworks that focus on improving the maturity of software processes, a goal-oriented agile SPI framework would focus on improving the effectiveness of software processes with respective to stated objectives. In future work, we intend to use the method fragment evaluation framework as a part of an agile SPI framework, in which the local improvements (improving method fragments) provide the basis for global optimization (improved overall process).

Limitations of the framework include reliance on human judgment during qualitative evaluation, and when there is no adequate evidential data. For instance, during the model refinement and initialization phase, the correct identification of significant situational factors is crucial. This issue has been widely studied in

² An Eclipse-based tool for modeling and evaluation of i^* goal models, available on-line at: <https://se.cs.toronto.edu/trac/ome>

situational method engineering [2]. In this framework we have tried to propose a simple approach for modeling project situation factors. Another limitation of this framework is the value propagation and resolution phase, in which the manual resolution of value bags is subject to personal judgments. Of course the experience of model evaluator is important here, but we hope to reduce the risks of such subjective decisions by enriching the evidence base. However, the evidence base is unlikely to cover all possible situations as there might be always some short-cuts to achieve goals, or some hidden factors that impede certain objectives of method fragments, regardless of their stated requisites. Nevertheless, the proposed framework can potentially help process managers make better-informed decisions based on the growing body of empirical evidence.

Appendix: List of Referenced Studies

Code	Approach	Reference
S1	web-based survey	Begel, A., & Nagappan, N. (2007). Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. First International Symposium on Empirical Software Engineering and Measurement, 2007 (ESEM 2007)., 255-264.
S2	Observatory	Sutherland, J., Viktorov, A., Blount, J., & Puntikov, N. (2007). Distributed Scrum: Agile Project Management with Outsourced Development Teams. 40th Annual Hawaii International Conference on System Sciences, 274a.
S3	Experience Report	Given, P. (2006) Scrum in a Fixed-Date Environment. accessed via: http://www.scrumalliance.org/articles/29-scrum-in-a-fixeddate-environment , in May, 2009
S8	Experience Report	Moore, R., Reff, K., Graham, J., & Hackerson, B. (2007). Scrum at a Fortune 500 Manufacturing Company. <i>Agile</i> , 175 - 180.
S12	Experience Report	Berczuk, S. (2007a). Back to Basics: The Role of Agile Principles in Success with an Distributed Scrum Team. <i>Agile</i> , 382-388.
S15	Ethnographic study	Chong, J., & Hurlbutt, T. (2007). The Social Dynamics of Pair Programming. Proceedings of the 29th international conference on Software Engineering, 354-363, IEEE Computer Society.
S17	Controlled Experiment	Sfetsos, P., Stamelos, I., Angelis, L., & Deligiannis, I. (2009). An experimental investigation of personality types impact on pair effectiveness in pair programming. In <i>Empirical Software Engineering</i> , 14(2), 187-226.
S30	Experience Report	O'Donnell, M. J., & Richardson, I. (2008). Problems Encountered When Implementing Agile Methods in a Very Small Company. (Ed.).In: <i>Software Process Improvement</i> (pp. 13-24). Springer

References

1. S. Brinkkemper, Method engineering: engineering of information systems development methods and tools. In *Information and Software Technology*, 38(4) p. 275-280 (1996)
2. A.F. Harmsen, *Situational Method Engineering*, Utrecht, Moret Ernst & Young Management Consultants (1997)
3. J. Ralyté and C. Rolland, An Assembly Process Model for Method Engineering, in *Advanced Information Systems Engineering*, p. 267-283 (2001)

4. M. Saeki. CAME: The first step to automated method engineering. In Workshop on Process Engineering for Object-Oriented and Component-Based Development. Anaheim, CA (2003)
5. B. Henderson-Sellers, Method engineering for OO systems development. In Communications of the ACM, 46(10) p. 73-78 (2003)
6. M. Bajec, D. Vavpotic, and M. Krisper, Practice-driven approach for creating project-specific software development methods. In Information and Software Technology, 49(4) p. 345-365 (2007)
7. W. Linda and K. Mark, Software project risks and their effect on outcomes. In Communications of ACM, 47(4) p. 68-73 (2004)
8. K. Beck, M. Beedle, A.V. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R.C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas (2001) Manifesto for Agile Software Development.
9. B.A. Kitchenham, T. Dyba, and M. Jorgensen, Evidence-Based Software Engineering, in Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society 2004)
10. K. Schwaber and M. Beedle, Agile Software Development with Scrum, Prentice Hall PTR. 158 (2001)
11. J. Ralyté, R. Deneckère, and C. Rolland, Towards a Generic Model for Situational Method Engineering, in Advanced Information Systems Engineering. p. 1029-10291 (2003)
12. D. Firesmith, Open Process Framework (OPF), accessible via: date accessed: November 2009
13. E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In Proceedings of the Third IEEE International Symposium on Requirements Engineering: (received "Most Influential Paper After 10 Years Award" at RE'07) (1997)
14. E. Yu and J. Mylopoulos. Understanding "why" in software process modelling, analysis, and design. In Proceedings of the 16th international conference on Software engineering. Sorrento, Italy: IEEE Computer Society Press (1994)
15. K.V. Slooten and S. Brinkkemper. A Method Engineering Approach to Information Systems Development. In Proceedings of the IFIP WG8.1 Working Conference on Information System Development Process: North-Holland Publishing Co. (1993)
16. C. Coulin, D. Zowghi, and A.-E.-K. Sahraoui, A situational method engineering approach to requirements elicitation workshops in the software development process In Software Process Improvement and Practice, 11(5) p. 451-464 (2006)
17. J. Horkoff and E. Yu. Using the i* Evaluation Procedure for Model Analysis and Quality Improvement presentation. In Second International Workshop on i* / Tropos. University College London, London UK (2005)
18. J. Horkoff and E. Yu. A Qualitative, Interactive Evaluation Procedure for Goal- and Agent-Oriented Models. In Proceedings of CEUR Workshop in CAiSE,09 (2009)
19. C. Gonzalez-Perez, P. Giorgini, and B. Henderson-Sellers. Method Construction by Goal Analysis. In Proceedings of Int. Conf. on Information System Development: Springer US (2007)
20. J. Ralyté, M.A. Jeusfeld, P. Backlund, H. Kühn, and N. Arni-Bloch, A Knowledge-based Approach to Manage Information Systems Interoperability. In Information Systems, Special issue on Advances in Data and Service Integration, 33(7-8) p. 754-784 (2008)
21. P.J. Ågerfalk and B. Fitzgerald, Exploring the concept of method rationale: A conceptual tool for method tailoring. In Advanced Topics in Database Research p. 63-78 (2006)
22. M. Rossi, B. Ramesh, K. Lyytinen, and J. Tolvanen, Managing Evolutionary Method Engineering by Method Rationale. In Rationale. Journal of the Association for Information Systems, 5(9) p. 356-391 (2004)
23. I. Aaen, A. Borjesson, and L. Mathiassen, SPI agility: How to navigate improvement projects. In Software Process: Improvement and Practice, 12(3) p. 267-281 (2007)