Incremental Method Enactment for Computer Aided Software Engineering Tools

Kevin Vlaanderen, Geurt van Tuijl, Sjaak Brinkkemper, and Slinger Jansen

Utrecht University,
Department of Information and Computing Sciences,
P.O. Box 80.007,
3508 TA, Utrecht, the Netherlands,
{k.vlaanderen,g.j.vantuijl,s.brinkkemper,s.jansen}@uu.nl

Abstract. In most cases, enactment is the most resource consuming aspect of process improvement, as large process changes are put into practice. Problems that typically are encountered include ineffective process changes, resistance from employees, and unclarity about the advantages of the new process. These problems can be avoided by incrementally implementing process changes, especially if the enactment is supported by process management tools that immediately change the processes and workflow in information systems. In this paper, we explain and demonstrate the concept of incremental method enactment for CASE tools. The concept is evaluated through a prototype, which is assessed by industry experts. The results of this study point give direction towards the further development of incremental method enactment.

Keywords: Method Engineering, Method Evolution, CASE tools, CAME tools, Enactment

1 Introduction

The methods that organizations employ are subject to constant change. Implementing changes to existing processes requires a lot of time and effort, and its success is prone to many factors, especially when the changes are significant. Issues such as resistance to change, lack of evidence, and resource constraints cause many process improvement efforts to fail [2]. Not surprisingly, this implementation or enactment of processes and process changes is an import subject within the Software Process Improvement domain [10,11,4].

Much research has been performed related to determining *what* changes should be introduced in order to improve process, and thus software, quality. This has resulted in a wide range of frameworks including CMMI [6], and SPICE [9]. Such frameworks need the addition of research on *how* changes should be introduced. The success of process improvement is highly dependent on contextual factors that go beyond the improvement content. In their analysis of SPI success in small to medium enterprises, [20] have determined multiple success factors, two of which are particularly important in the context of this research:

– ”Guide the improvement programme, following a systematic and coherent initiative by means of specific procedures and combining different approaches. This procedure must follow an iterative and incremental approach (prioritize the improvement points defined by the organization) that allows a continuous adoption of improvement practices.” [20]
– ”Tackle the problem of improvement from the technical perspective.” [20]

The first of these two factors or guidelines lays at the core of what is called *incremental method evolution* or incremental process improvement, which deals with the step-wise improvement of process development processes [26]. Incremental method evolution focuses on delivering tools and techniques that enable small, local changes to a method with the goal to minimize the overall disturbances on a business process and to allow for an iterative and 'natural' approach to process improvement. The second guideline is important because, with the advent of modern technology, automated process and computer-supported process execution becomes more and more feasible.

Most software organizations rely on several CASE tools during the development of software products. Such tooling often requires significant configuration and maintenance during its lifecycle, and adaptations to it can thus cause significant resistance among system administrators and users alike. Incremental method enactment as we use it here, aids the transformation of a method and the implemented process by alleviating some of the issues that come in play when they are changed, mainly with regard to tooling maintenance.

These challenges form the basis for the research question answered in this paper, which is formulated as *How can the enactment of incremental method changes in CASE tools be facilitated?*

The research presented in this paper elaborates on the enactment of changes in situational methods, from a technical perspective, in a dynamic environment. We propose a mechanism for synchronizing the evolution of situational methods and the tools supporting them throughout the lifecycle of a method. This solution, referred to as incremental method enactment, describes the process of automatic changes in software engineering tools according to incremental changes in methods and their instances. The mechanism is demonstrated by a proof of concept. Process Deliverable Diagrams are used for modeling multiple method increments from a process and work product perspective. These diagrams are translated automatically into the datamodel of a CASE tool in order to update both the workflow of the tool as well as the work products facilitated by it. The proposal is validated through multiple interviews with industry experts.

2 Incremental Method Enactment

The research community has long recognized that the idea of fully standardized and stable methods across multiple organizations is a utopia [5,19]. Organizations employ multiple, often interdependent methods that vary in their amount of formalization and quality. The environment in which the organizations operate changes, and the internal methods need to change accordingly. Existing information systems development methods are often too generic and cannot be followed [15]. They need to be tuned to a specific situation for the project at hand [14,22,3] by incorporating the uniqueness of a specific project, which lies at the core of situational method engineering (SME). SME denotes a group of techniques that allow organizations to develop tailor made methods tuned to their specific situation and is defined as the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems [3]. During the last several years, several modularization constructs have been proposed for situational method engineering. Although these constructs have many aspects in common, some essential differences exist. Extensive comparisons of these constructs were performed by [7], [1] and [8]. Each of these techniques focuses on the construction and/or adaptation of methods based on (parts of) previous methods. Instead of spending effort on adapting the organization to a standardized process, it is spent on thoroughly analyzing the organization and developing a process that better fits the organizational structure, capabilities, and habits. Theory predicts that the implementation of such situational methods has a higher chance of success by lowering the barrier for introducing changes.

The concept method enactment, depending on the research line, has gained much attention in the software process improvement (SPI) and situational method engineering (SME) domains. In general, enactment refers to the instantiation of a process model or a method to a real-life scenario, in other words 'putting it into action'. As was already recognized by [10], enactment deals not only with the implementation of a method through tooling, but also with training, controlling, etc. of human actors that perform the process. From the viewpoint of situational method engineering, method enactment represents the final stage of the construction or the tailoring of a method for a specific project at hand. It completes a round-trip, starting from an analysis of the current situation and ending at an improvement of that situation.

The Structure of Method Increments Incremental method enactment implies an iterative approach to the development and instantiation of a process. As the term implies, it is based on the notion of a method increment. The basic components of a method increment are generally a description of the changes involved, both in terms of activities as well as deliverables, and possibly a description of the goal and experiences. The latter part is often described as the rationale. Based on the notion of design rationale, [21] informally define method rationale as information about decisions that lead to a certain meta-model. From a more structural viewpoint, a method increment has been defined as a collection of method fragments that have been introduced in a method between two points in time [29]. Conceptually, it represents a coherent set of changes to an existing method in order to facilitate evolutionary changes.

Continuing in this line, we adopt the notion of method increments and its proposed modeling approach. Essentially, method increments are modeled in the form of Process-Deliverable Diagrams (PDDs) [28], where each consecutive

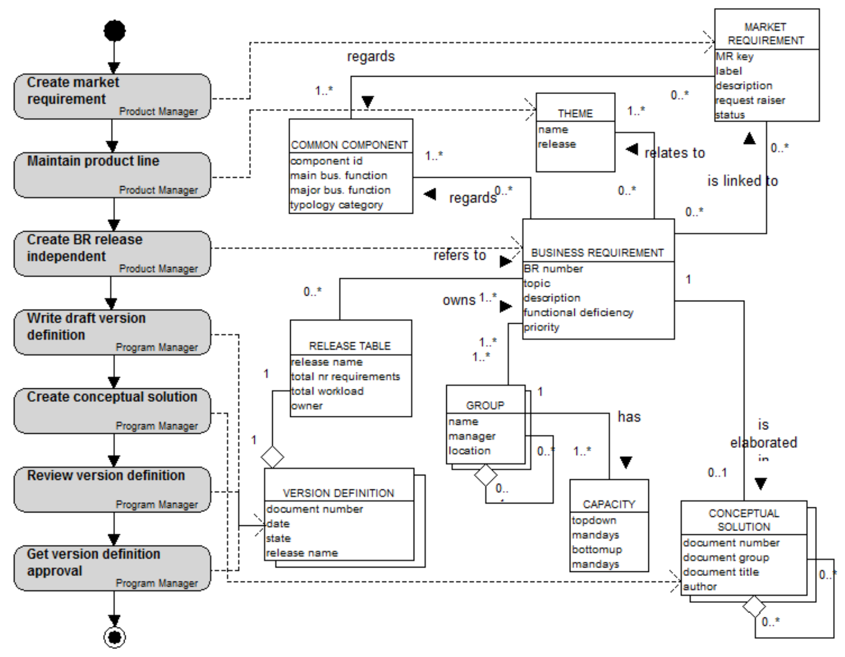


Fig. 1. Example Process Deliverable Diagram; Increment 4 of the Case Study

model describes the state of a method fragment at a specific point in time [29]. The evolution of a method fragment is therefore described by a series of PDDs, thus creating something that resembles a storyline.

In the context of method engineering, and described in terms of the Meta Object Facility (MOF) levels [12], the process part of a PDD is at level M1 (class level), while the product part remains one level higher at M2 (metaclass level), as shown by [17]. During the execution of a process, both parts are instantiated, resulting in specific activities (M0 level) and models such as requirements or use case diagrams (M1 level). CASE tools are designed to support the instantiations, thus acting at the same meta-levels as a PDD. Figure 1 demonstrates this by showing one of the PDDs that have been used during this study. This specific example describes the process of capturing market requirements and transforming them into business requirements that end up in a specific release.

2.1 The Process of Incremental Method Enactment

Both the meta-modeling activity as well as the modeling activity can be performed in a variety of ways and supported by a variety of tools, depending on the specific context and processes. For instance, numerous tools have been developed for supporting the requirements engineering activity. Each tool provides more or less freedom to the user in terms of modeling paradigms that can be used.

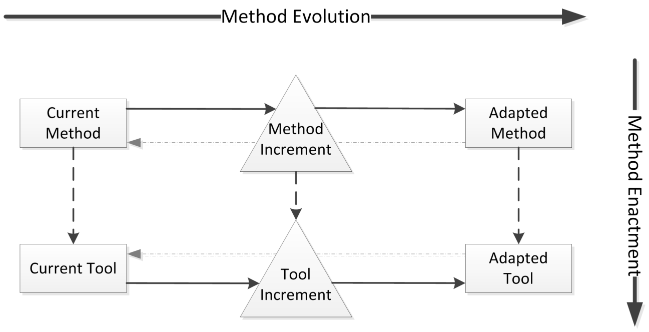


Fig. 2. Conceptual Illustration of Incremental Method Enactment

On the other hand, the meta-modeling activity can be supported by a variety of Computer Aided Method Engineering (CAME) tools, supporting activities such as the creation, modification, and combination of method fragments. In some cases, such as in the case of MetaEdit+ [18], both worlds are combined, allowing method engineers to define a modeling environment that can be instantiated and in turn be used by, for example, the software engineer. This by itself can be seen as a form of method instantiation, as it involves the transformation of the meta-model level to the model-level. However, true situational method engineering solutions should not force engineers (on either level) to specific tools. A typical software engineering process often involves multiple models and is subject to the capabilities and the context of the organization.

Based on the above, we can say that incremental method enactment, from a technical perspective, is a continuous adaptation of the meta-model combined with a continuous synchronization of the meta-model and the CASE tool, as illustrated by Figure 2. The goal is to facilitate changes in the software engineer's process by correctly transforming a CASE tool's configuration to match the method's meta-model. Changes to the activity side result in an altered workflow, while changes to the deliverable side result in altered work products. The challenge is then to support the enactment of both the process as well as the deliverable aspect of a method fragment while taking into account the specific organizational context.

This can be done through an intermediary tool, such as demonstrated in Figure 3. Such a tool extends the functionality of the CAME environment by linking the meta-meta-model (i.e. the modeling language used for creating method fragments) to a specific CASE tool's interface. The constructs used within the instantiation of this meta-meta-model are translated into the constructs that are used within the CASE tool. On each consecutive change to the method, the exact adaptations are calculated and propagated to the CASE tool, enabling a smooth transitions from one method version to the next.

Such a translation is highly dependent on both modeling environments, and defining a modeling paradigm agnostic approach lies beyond the scope of this

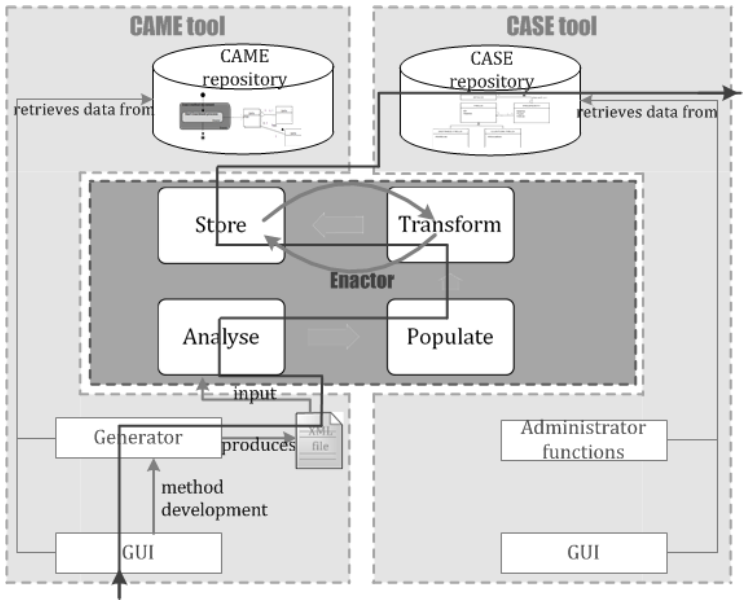


Fig. 3. Overview of the Enactment Mechanism

paper. However, we demonstrate and evaluate the potential workings of such an approach through a prototype in the next chapter.

3 Prototype

To demonstrate the incremental method enactment approach, we have built a prototype enactment mechanism. This mechanism is used to evolve a live CASE tool according to method changes. The prototype allows us to evaluate both the requirements for implementation as well as the mechanism itself.

3.1 Selection of Tools

On the one hand, the CAME tool needs to satisfy several technical requirements. The CAME tool should be able to create valid increments according to the chosen modeling paradigm, in this case the PDD. Another requirements is that the CAME tool is capable of transforming a graphical representation to structured textual data. Finally, the CAME tool should include the possibility of invoking the enactment mechanism that is responsible for pushing the changes to the CASE tool.

Through an analysis of previous literature, we have identified seven available CAME tools, namely MERET [16], MethodBase [23], Decamerone [14], MENTOR [24], MetaEdit+ [18], MERU [13] and Method Editor [22]. Out of these tools, MetaEdit+ is the most actively developed. In earlier research, a metamodel has been created for Process-Deliverable Diagrams in MetaEdit+ (see [27]). This metamodel is used to create valid PDDs through the MetaEdit+ interface. Moreover, MetaEdit+ is able to transform a graphical fragment to structured text and allows for the execution of external functionality.

Similarly, the CASE tool is subject to a set of high-level requirements as well. Primarily, the CASE tool needs to include a means to support the process side as well as the deliverable side of a PDD. This can be translated to the ability of workflow management in addition to work product management. Furthermore, the configuration of the CASE tool should be accessible and modifiable through an external interface. This interface can be provided in terms of an Application Programming Interface (API), a manual function such as the ability to import external configuration files, or direct access to the application database. In addition, the CASE tool needs to be rather mature. CASE tools that were too superficial or too concise (such as small mock-ups or simple programming classes) were eliminated from the list. Finally, The selected CASE tool should be well-known to the IT-industry. We marked a CASE tool as 'well-known' if we found several prominent companies (i.e. customers of the CASE tool) that are using the CASE tool.

For the selection of an appropriate CASE tool, we have created a list of available CASE tools, based on manual research on the internet. In total, we found over 50 usable CASE tools. After the selection procedure, we chose to use Jira[1] during the development of the prototype. Jira supports the definition of detailed workflows, enabling the implementation of the activity side of PDDs. The tool is flexible and very configurable. In addition, it is a widely used tool, enhancing the practical relevance of this research.

3.2 Mapping Increments to the CASE tool's Configuration

After selecting the appropriate tools, we created a mapping that allows us to convert the data of a method fragment to the CASE tool's configuration model. Figure 4 depicts this mapping.

On the left-hand side of Figure 4, a simplified fragment in the form of a PDD is shown, encircled with a dotted line. On the right-hand side of Figure 4, the essential elements of the CASE tool's datamodel are shown. The lines link the method fragment meta-model to the datamodel. Table 1 provides a textual explanation of the relationship between both fragments.

The diagram name of the fragment that a user creates is used to create an issue type in Jira. For instance, if you create a first version of a fragment that is entitled 'Requirement management process', this diagram name will be used to create an issue type in Jira. All the data that is entered in Jira underneath this issue type belongs to that specific fragment. When an updated version of the fragment is created, the existing issue type will not be affected. Issue types can be used in different projects or in one project only.

[1] http://www.atlassian.com/software/jira/

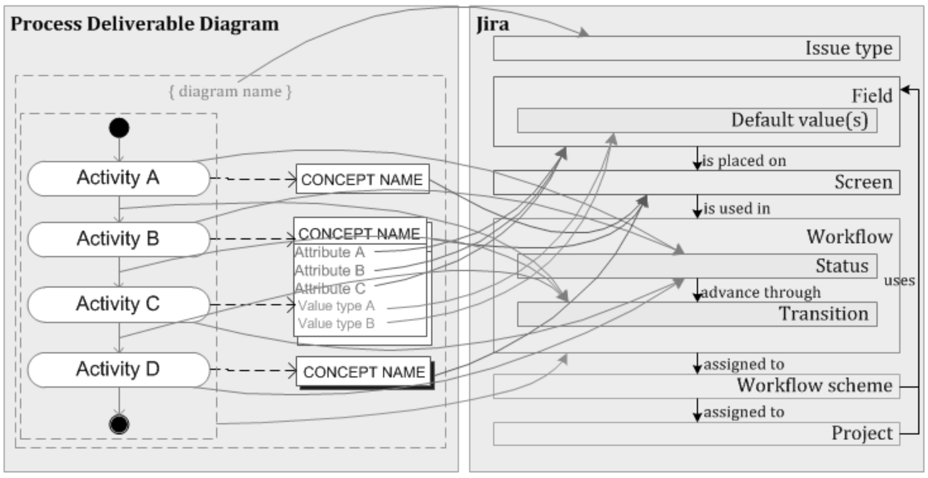


Fig. 4. Mapping of PDD and the CASE tool datamodel.

The complete process side of the fragment is used to create a workflow in Jira. A workflow in Jira is only represented by a name and the entire workflow is based on an XML structure. A workflow in Jira is similar to the process side that is modeled in a PDD. A workflow exists of a status and transitions. A status is a step in the workflow through which you advance while working in Jira. For instance, if you complete activity A in the workflow and mark this as complete, you advance to the next status (i.e. the next activity). Advancing through statuses in Jira requires transitions. Transitions are moments in the workflow in which you can complete one activity and continue with the next activity.

Table 1. Mapping of Method Fragments to Jira

#	Fragment	Relationship	Jira tool fragment	Usage domain
1	Diagram name	is used to create an	Issue type	per project
2	Process side	will produce a	Workflow	per project
3	Activity	is used to create a	Status	in a workflow
4	Transition	is used to create a	Transition	in a workflow
5	Concept	is used to create a	Screen	per project
6	Attribute	is used to create a	Field	per project
7	Value type	will be used as (a)	Default value(s)	per field

An activity within a PDD is used to create a status in Jira. The status in Jira is a direct copy of the activity as described in the PDD. Statuses in Jira are used within workflows. They are used to indicate at which point the user currently is in the entire workflow, similar to a PDD. Each status is linked to a screen so that a status can present information to the user.

A transition in a PDD is the same as a transition in a workflow in Jira. They are used to close an activity a user is currently dealing with and to continue with the next activity in the workflow. The name of the transition is based on the upcoming activity. For example, if you are at 'Gather requirements' in the workflow, the transition name will be 'Write draft version definition' if this is the upcoming activity.

The name of a concept is used to create a screen in Jira. For instance, if you have a concept called REQUIREMENT, a screen is created with the name REQUIREMENT. A screen contains fields that are based on attributes of the concept. In the event that a concept does not contain attributes, the screen does not have any input fields. Although this seems to make little sense, the screen is linked to a status that can contain many other functions such as attaching a document or adding comments to a particular status. In that case having a screen can still be very useful even though it contains no fields.

An attribute of a concept within a PDD is used to create a field with the same name. The field that is created will be placed on the screen of the respective concept. For instance, if a concept called REQUIREMENT has an attribute called code, a screen is created (namely REQUIREMENT) which has a field called code. If an attribute has attributes, these attributes will be used as default values of the respective field. For instance, if the attribute code has attributes such as G1, G2 or G3, a select box field is created that contains the values G1, G2 of G3. These values can be selected from a drop down box for instance.

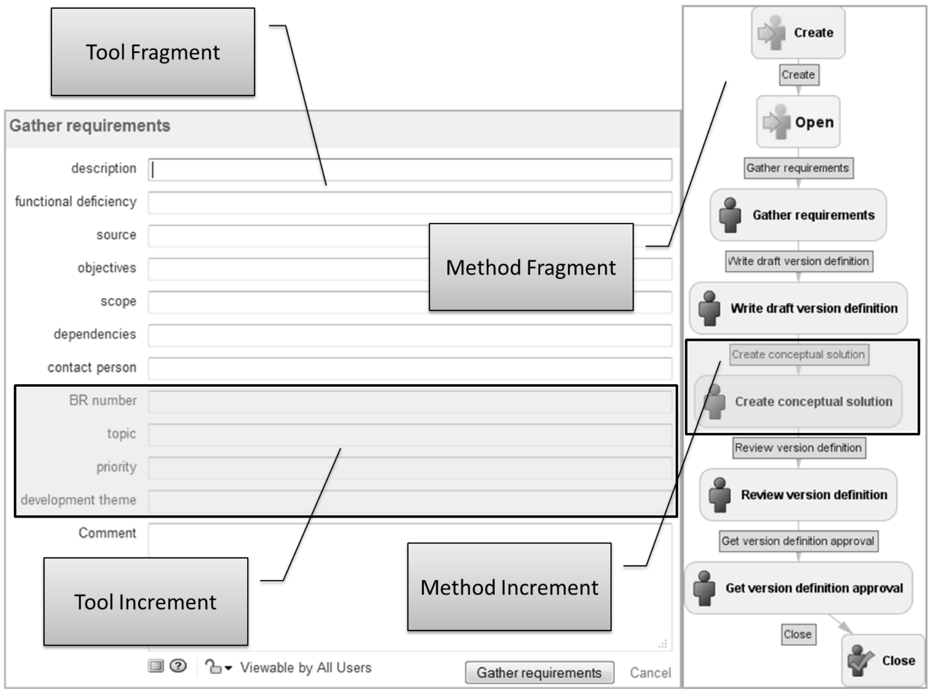
3.3 The Enactment Mechanism

The enactment mechanism facilitates the translation from a PDD to the internal datamodel of our CASE tool. After a fragment has been created in MetaEdit+ (see Fig. 3 for an example), a MERL script (an internal scripting language) exports the fragment to an external tool. This tool extracts the relevant meta-data from the fragment, including the fragment name and its version, and converts the fragment to an intermediate structure based on the PDD meta-model as defined by [29].

Once all relevant data has been extracted, it is shown in a user friendly manner, in order to let the user verify the consistency of the data. Once the data is verified, the enactment process is triggered. For each relevant database table of the CASE tool, three objects are created based on the Model-View-Controller (MVC) design pattern; a Value Object (VO), a Data Access Object (DAO), and a Controller Object (CO). These objects form the connection between the MetaEdit+ datastructure and the CASE tool datastructure.

Through a set of prepared SQL statements, the data fragment is then stored in the CASE tool configuration database. Prepared statements are used to increase safety and avoid SQL injection (i.e. attacking the database deliberately through query manipulation). We used PDO in conjunction with MySQL 5.0 because this allows us to abort the entire querying execution process when a single query fails. In the case a query fails at any point in the query execution process, the whole query execution process can be rolled back without leaving any data behind. In that case not a single change is made to the data in the database of the CASE tool, avoiding contamination and perhaps a broken system.

of each fragment can be found in [30].


Tool Fragment
Method Fragment
Tool Increment
Method Increment
Create
Open
Gather requirements
Write draft version definition
Create conceptual solution
Review version definition
Get version definition approval
Close

Gather requirements

description
functional deficiency
source
objectives
scope
dependencies
contact person
BR number
topic
priority
development theme
Comment

Viewable by All Users
Gather requirements
Cancel

Fig. 5. Adapted Method Fragment in the CASE tool.

Figure 5 shows the method increment and tool increment based on increment #4 in [30]. On the left side of Figure 5, the fields for the respective concepts of increment #4 are shown. The fields in Jira are based on the attributes of the concepts REQUIREMENTS DOCUMENT and REQUIREMENT. On the right hand side of Figure 5, the workflow is shown that is in accordance with the workflow in fragment #4. The shaded areas denote the method and tool increment.

4 Evaluation

The research presented in this paper demonstrates an early attempt at supporting the technical enactment of continuous method improvements. To determine which aspects should receive more attention in future research, we used the prototype to evaluate the approach. In this section, we summarize both a technical evaluationan as well as an expert evaluation.

4.1 Technical Evaluation

We modeled five different increments that we used to evaluate the enactment mechanism. To emphasize the notion of mimicking a real-life setting, we selected the base method and the first four method increments from a previously conducted case study [30] at a vendor of ERP software. The increments show the evolution of a requirement management method that advances from a very simple method to a more elaborate approach. The increments, along with a textual elaboration of each increment, can be found in [30]. An overview of the method increments used during this study is found in table 2.

Table 2. Overview of Method Increments

#	**Goal**	**Date**
0	Introduction requirements document	1994
1	Introduction design document	1996
2	Introduction version definition	1998, May
3	Introduction conceptual solution	1998, November
4	Introduction requirements database, division market and business requirements, and introduction of product families	1999, May

During the development of the prototype, we encountered some issues that still inhibit a fully automated approach to method enactment.

Standardization of Process Deliverable Diagrams. In principle, the PDD format has been clearly defined and formalized [28]. However, in practice we encountered many PDDs with different appearances and rules. Unfortunately, unambiguous modeling rules and notations still lack which often caused trouble when automating the data of a PDD through a mechanism.

Usage of open, closed and sub activities. Open, closed and subactivities in a fragment indicate different levels of aggregation within the process. However, the ability to implement these aggregation levels highly depends on the CASE tool, and many don't distinguish between different levels of activities, e.g. phases, steps; each activity forms a step in the workflow. Because the work-flow possibilities in Jira lack comprehensive support, each activity in a fragment is seen as a regular step in the workflow.

Usage of forks, joins and decisions. The use of forks, joins and decisions in a CASE tool is limited, caused by the amount of default features that are available in a CASE tool. In our case, we could use decisions and/or logical flows in our fragment only because Jira lacks comprehensive support for these special types of activity nodes.

Generalizations, associations and aggregations. In our mapping, we did not find a way through which we could make sound use of generalizations, associations and aggregations. The implementation of these types of relationships is unknown, also caused by the available amount of features in the CASE tool. Jira, in its current form, was not able to provide sufficient support for these types of special relationships.

4.2 Expert Evaluation

We conducted 5 semi-structured interviews at different medium/large-scale organizations. We interviewed 5 product managers from different IT branches in order to evaluate the rationale behind the approach, the feasibility of the enactment mechanism, and the required adjustments. We posed a total of 14 questions. 7 of these were aimed at eliciting information directly related to the companies' current situation. The other 7 focused on the evaluation of the enactment mechanism. In between, a demo was shown to the product manager in the form of a video that showed the mechanism in action[2]. During the video, a verbal explanation was provided to make things clearer to the product manager. At the end of each interview, time was available to document any kind of other information that was provided by the product manager, i.e. remaining concerns and/or opinions.

Each of the interviewees agreed on the fact that process adaptation is an important and continuous activity throughout the organization or department. Each organization struggled in the beginning with their business process(es), and each company reinvented the wheel numerous times before a standard process was in place. 4 out of 5 companies indeed had process models in place, mostly at a high level of detail. One company did not have any model in place due to its small size. Over time, each company shifted from having a very detailed process model to a more high-level process.

As indicated by the interviewees, one of the major issues with the current approach is its rigidity. A successful enactment mechanism needs to be flexible, supporting non-linear, complex methods, and allowing exceptions. This includes the support for decisions, multiple stakeholders, and a flexible stance towards the relation between the process model and reality. Most interviewees agreed that any enactment approach that forces people to work in a specific way would be met with much resistance, which is a result that is in line with most recent SPI [2,25].

[2] The video can be downloaded from `http://www.staff.science.uu.nl/~vlaan107/enactment_demo.wmv`

Besides issues of resistance and rigidity, the interviewees commented on the fact that the current approach does not deal with all possible process model changes. It focuses mainly on the addition of steps and/or deliverables, while leaving out the deletion of process elements. This caused concern related to the reliability of the approach and the availability of data.

5 Conclusions and Further Research

In this research, we attempted to provide an answer to the following research question:

> How can method increments be employed in software engineering tools and allow for incremental method enactment?

We established an alternative viewpoint on how process models can be useful within organizations. The making of process improvement models tends to absorb a lot of time and resources while they are rarely used. In this research, we demonstrated how such models can facilitate the enactment of processes by using them to adjust CASE tools.

The enactment mechanism that we developed analyzes and validates method fragments. It extracts all relevant knowledge, which is then stored in a structured manner, after which the data is converted to a standard that is compliant with the CASE tool. Finally, all the elicited data from the fragment is stored in the repository of the CASE tool. When the four phases are executed successfully in a chronological order, the configuration of the corresponding software engineering tool is automatically tailored to the method fragment. When the enactment mechanism is invoked after each created fragment, incremental method enactment is achievable in the corresponding CASE tool.

Although we used a prototype to demonstrate and evaluate the mechanism and the concepts behind it, it should be noted that the implementation was specific to Jira. Still, the mechanism was perceived as potentially useful by practitioners, and it is a first step towards enactment support for process evolution. However, the current mechanism was constrained by using a linear process flow only. In order for the mechanism to become a useful tool, the current mechanism needs more functionality by incorporating workflow support for iterative processes, decisions, joins and forks, although this is partially influenced by the selected CASE tool. The selected CASE tool should at least provide support for the process side (i.e. workflow support) and features to operationalize the deliverable side of a fragment.

The interviewees placed strong emphasis on ease of use, advanced workflow support (including decisions, multiple roles, and iterative processes), and the ability to show the transparency of (internal) processes towards the customer(s) using the approach. Therefore, during next iterations of the design process, we will focus on these aspects.

References

1. Å gerfalk, P.J., Brinkkemper, S., Gonzalez-Perez, C., Henderson-Sellers, B., Karlsson, F., Kelly, S., Ralyté, J.: Modularization Constructs in Method Engineering: Towards Common Ground? Situational Method Engineering: Fundamentals and Experiences 244, 359–368 (2007)
2. Baddoo, N.: De-motivators for software process improvement: an analysis of practitioners' views. Journal of Systems and Software 66, 23–33 (Apr 2003)
3. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology 38(4), 275–280 (1996)
4. vom Brocke, J., Sinnl, T.: Culture in business process management: a literature review. Business Process Management Journal 17(2), 357–378 (2011)
5. Brooks, F.: No Silver Bullet. IEEE Computer 20(4), 10–19 (1987)
6. CMMI Product Team: Capability Maturity Model Integration (CMMI). Tech. Rep. December 2001, Carnegie Mellon Software Engineering Institute, Pittsburgh (2002)
7. Cossentino, M., Gaglio, S., Henderson-Sellers, B., Seidita, V.: A metamodelling-based approach for method fragment comparison. In: Proceedings of the 11th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD06) (2006)
8. Deneckère, R., Iacovelli, A., Kornyshova, E., Souveyet, C.: From Method Fragments to Method Services. In: Halpin, T., Proper, H., Krogstie, J. (eds.) Proceedings of EMMSAD'08. Montpellier, France (2008)
9. El Emam, K.: SPICE: The theory and practice of software process improvement and capability determination. IEEE Computer Society Press, Los Alamitos, CA, USA (1997)
10. Feiler, P., Humphrey, W.: Software process development and enactment: Concepts and definitions. In: International Conference on the Software Process. Software Engineering Institute, Pittsburgh, PA (1993)
11. Gonzalez-Perez, C., Henderson-Sellers, B.: A work product pool approach to methodology specification and enactment. Journal of Systems and Software 81(8), 1288–1305 (Aug 2008)
12. Group, O.M.: Meta Object Facility (MOF^{TM}). Tech. rep., Object Management Group (2011)
13. Gupta, D., Prakash, N.: Engineering Methods from Method Requirements Specifications. Requirements Engineering 6(3), 135–160 (Oct 2001)
14. Harmsen, F., Brinkkemper, S.: Design and implementation of a method base management system for a situational CASE environment. In: Proceedings of the Second Asia-Pacific Software Engineering Conference (APSEC'95). p. 430. IEEE Computer Society, Washington, DC, USA (1995)
15. Harmsen, F., Brinkkemper, S., Oei, J.L.H.: Situational method engineering for informational system project approaches. In: Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle. pp. 169–194. Elsevier Science Inc., New York, NY, USA (1994)
16. Heym, M., Osterle, H.: A semantic data model for methodology engineering. In: Computer-Aided Software Engineering, 1992. Proceedings., Fifth International Workshop on. pp. 142–155. IEEE (1992)
17. Jeusfeld, M.A.: A Deductive View on Process-Data Diagrams. In: Proceedings of the IFIP WG8.1 Working Conference on Method Engineering. Paris, France (2011)

18. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+ A fully configurable Multi-User and Multi-tool CASE and CAME Environment. In: Proceedings of the Conference on Advanced Information Systems Engineering. pp. 1–21 (1996)
19. Malouin, J., Landry, M.: The mirage of universal methods in systems design. Journal of applied systems analysis (1983)
20. Pino, F.J., García, F., Piattini, M.: Software process improvement in small and medium software enterprises: a systematic review. Software Quality Journal 16(2), 237–261 (Nov 2008)
21. Rossi, M., Tolvanen, J.: Method rationale in method engineering. In: Proceedings of the Hawaii International Conference on System Sciences. pp. 1–10. Hawaii (2000)
22. Saeki, M.: CAME: The first step to automated method engineering. In: Crocker, R., Steele Jr., G.L. (eds.) Proceedings of the Workshop on Process Engineering for Object-Oriented and Component-Based Development. pp. 7–18. ACM, Anaheim, California, USA (2003)
23. Saeki, M., Iguchi, K.: A meta-model for representing software specification & design methods. In: Proceedings of the IFIP WG8.1 Working Conference on Information System Development Process (1993)
24. Si-Said, S., Rolland, C., Grosz, G.: MENTOR: a computer aided requirements engineering environment. In: Advanced Information Systems Engineering. pp. 22–43. Springer (1996)
25. Sulayman, M., Urquhart, C., Mendes, E., Seidel, S.: Software process improvement success factors for small and medium Web companies: A qualitative study. Information and Software Technology 54(5), 500–479 (Jan 2012)
26. Tolvanen, J.P.: Incremental method engineering with modeling tools: theoretical principles and empirical evidence. University of Jyväskylä (1998)
27. Vlaanderen, K., van de Weerd, I., Brinkkemper, S.: On the Design of a Knowledge Management System for Incremental Process Improvement for Software Product Management. International Journal of Information System Modelling and Design (Selected papers of ME'11), accepted for publication (2012)
28. van de Weerd, I., Brinkkemper, S.: Meta-modeling for situational analysis and design methods. In: Handbook of Research on Modern Systems Analysis and Design Technologies and Applications, chap. III, p. 35. Information Science Publishing (2008)
29. van de Weerd, I., Brinkkemper, S., Versendaal, J.: Concepts for Incremental Method Evolution : Empirical Exploration and Validation in Requirements Management. In: International Conference on Advanced Information Systems Engineering. pp. 469–484. Springer (2007)
30. van de Weerd, I., Brinkkemper, S., Versendaal, J.: Incremental method evolution in global software product management: A retrospective case study. Information and Software Technology 52(7), 720–732 (Jul 2010)