Accepted for the 6th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'01), Interlaken, Switzerland, 4–5 June 2001.

# Method Configuration with Development Tracks and Generic Project Types

Fredrik Karlsson[1], Pär J. Ågerfalk[1], Anders Hjalmarsson[2]

Research Group VITS

[1] Dept. of Informatics (ESA)
Örebro University, SE-701 82 Örebro, Sweden
{fkn | pak}@esa.oru.se

[2] School of Business and Informatics (IDA)
University College of Borås, SE-501 90 Borås, Sweden
ahj@hb.se

## Abstract

The world of system development methods is changing as rigorous 'off-the-shelf' methods become more popular. The need for configuration of such methods in a structured way is increasing accordingly. In this paper, method configuration is considered as a particular kind of method engineering focusing on adaptation of a base method. We propose a method configuration process based on the concepts of Development Tracks and Generic Project Types. A Development Track is a pre-made ideal process configuration suitable for a delimited characteristic of a (type of) software artefact, or a (type of) software development project, or a combination thereof. Generic Project Types with different characteristics can be related to different Development Tracks and used as a base for a situational method. The aim of the proposed method configuration process is to ease the burden of configuring the base method in order to reach an appropriate situational method more efficiently.

## 1. INTRODUCTION

A great deal of method engineering (ME) research has been devoted to situational ME (e.g., van Slooten and Hodes, 1996; ter Hofstede and Verhoef, 1997; Harmsen, 1997; Brinkkemper *et al*., 1999; Rolland *et al*., 1999). The aim of frameworks for situational ME is usually to guide the process of selecting and integrating different method fragments into congruent and consistent methods relevant for the situation at hand (ter Hofstede and Verhoef, 1997). The situational method is often thought of as a combination of fragments from different methods, and the incentive for situational ME is thus a need to integrate the strengths of different methods (e.g., Brinkkemper *et al*., 1999; Goldkuhl *et al*., 1997). In this paper, we choose to focus on method configuration (MC) as a particular form of situational ME. MC means to adapt a particular method to various situated factors. The focus is thus on one method as a base for configuration rather than on a set of methods as a base for assembly. Nonetheless, during MC the base method might need to be enhanced with additional fragments from other methods as well. The important thing is that MC always takes a base method as a starting point. Consequently, method integration is considered peripheral to MC, whereas method adaptation is essential.

The current popularity of rigorous 'off-the-shelf' development methods and processes increases the need for MC. Process configuration is an integrated part of the Rational Unified Process (RUP) (Kruchten, 1999), which is a prominent example of such methods. However, the support for MC within RUP can be developed further, and this is the subject of our paper. In the paper we propose an MC process based on the concept of Development Tracks and

Generic Project Types. These concepts complement the concept of Development Case used within the RUP to describe project-specific process configurations. The results are not specific to the RUP even though the RUP has been used as base method in our MC research. Rather, the principles described will apply to a wide range of base methods since we will discuss general underpinning principles, not the RUP as such. Our approach to MC assumes a selection of a base method that is appropriate for the organization. It is, however, out of the scope of this paper to go into details of that.

The work was performed in cooperation with an industrial partner, Volvo Information Technology (Volvo IT for short), which is currently implementing the RUP as a corporate-wide development process. The problems faced in creating organization-specific (and even more detailed) Development Cases were the incentive for us to investigate a more efficient MC process than that which RUP currently has to offer. Specifically, our work has focused on the need for configuration of RUP for the development of Internet-based software artefacts (sometimes referred to as Web applications). We have found that many of the problems connected with the development of such artefacts are actually generic problems, applicable to many kinds of development situations and types of artefacts. That is why we chose to discuss Development Tracks and Generic Project Types, of which some, but not all, apply to Internet-based software artefacts.

The paper is organized as follows. Firstly, we argue the need for Development Tracks, and introduce an MC process tailored for working with the combination of a base method and a set of Development Tracks. Secondly, we discuss an additional concept important for MC and for the reuse of experiences during MC work—the concept of Generic Project Types. Thirdly, we present a tentative method configuration process that makes use of both Development Tracks and Generic Project Types. Finally, we conclude the work by commenting on the potential impact of our approach on ME and MC and the research process adopted for establishing its legitimacy. Before we begin, however, some thoughts on what a method is (or at least can be).

## 2. THE CONCEPT OF METHOD

Many scholars have tried to define the concept of method as it is used within the field of information systems (e.g., Checkland, 1981; Rumbaugh, 1995; Brinkkemper, 1996; Goldkuhl *et al*., 1998; Cronholm and Ågerfalk, 1999). Even though they all define 'method' slightly differently, there seems to be a common understanding that a method consists of three interrelated parts. Firstly, there is a process to guide the performance of activities. Secondly, there is some sort of notation used to document the results of the activities. Finally, there is a set of concepts used to describe the problem domain (i.e., modelling primitives) and the method itself.

A detailed view of these three basic constituents of methods is presented by Brinkkemper *et al*. (1999), who describe three dimensions for classifying method fragments referred to as *perspective*, *abstraction level*, and *layer of granularity*. The perspective dimension concerns the process *vs*. product view. The abstraction level defines whether a fragment is a conceptual fragment (e.g., a model or modelling primitive), or a technical fragment (e.g., a CASE-tool). Finally, any method fragment is said to reside on one of five different layers of granularity: method, stage, model, diagram, or concept.

It is important to recognize that the process view and the product view are interrelated. Products (models, diagrams, *et cetera*) will both be results of the process' activities and prerequisites for subsequent activities in the process. Therefore, the ordering of activities within a method is an important consideration in relation to produced results (products). In the next section's description of the concept of Development Track, the focus is on activities. The results of these activities are implicitly present, even though they are not explicitly mentioned throughout the text.

An important aspect of methods, stressed by, for example, Ågerfalk and Åhlgren (1999), is their argumentative dimension, i.e., the methods' rationale. Activities are performed for reasons. These purposes are important for MC and are used as the key to decide how to perform the different activities of the base method in particular configurations.

## 3. DEVELOPMENT TRACKS

A *Development Track* is an ideal process configuration suitable for a delimited *characteristic* of a (type of) software artefact, or a (type of) software development project, or a combination thereof. Carrying out a software development project means dealing with such characteristics, and hopefully turning them into an advantage (cf. Harmsen, 1997; Beck, 1999). Projects are complex phenomena and hence the performance of MC is inherently complex. Isolating one characteristic at a time is one way to reduce complexity. A characteristic can be thought of as an enumerated type (i.e., like an `enum` in C++), and we refer to the enumerated *values* as the characteristic's *dimension*. A Development Track can therefore be seen as a process configuration ideal for a specific value of a characteristic, selected from its dimension. Depending on divergences in the characteristics of a specific project, a choice is made from a range of such process configurations. A dimension is used to describe each characteristic.

An example of a characteristic that we have found when applying these ideas at Volvo IT is 'Degree of new functionality'. The characteristic's dimension consists of two values: 'Existing functionality with new GUI' and 'Completely new system'. This characteristic seems to be very frequent in Internet-based development, and especially the value 'Existing functionality with new GUI'. The characteristic can be formally described as follows:

```
Characteristic Degree_of_new_functionality (
   Existing_functionality_with_new_GUI,
   Completely_new_system
);
```

Development Tracks suitable for a specific characteristic would typically consist of special instructions on how to perform each activity in the base method with regard to the values of its dimension. In the 'Degree of new functionality' case, for example, activities related to database design are considered unnecessary if we are dealing with 'Existing functionality with new GUI'. Hence, the *performance* of these activities could be performed in a reduced form, or be skipped altogether, while other activities might be performed as suggested by the base method. The Development Track can be formally described as follows:

```
Development_Track Existing_functionality_with_new_GUI {
   ...
   Activity : Analyse_Behavior {Performance = Perform_as_is;};
   Activity : Design_Components {Performance = Perform_as_is;};
   Activity : Design_the_Database {Performance = Skip;};
   Activity : Refine_the_Architecture {Performance = Perform_reduced;};
   ...
};
```

Note that the complete description of a Development Track (i.e., without the ellipses in the example above) requires one statement about the performance of each and every activity prescribed in the base method.

### 3.1 Defining Development Tracks

Defining Development Tracks plays a central part in the proposed MC process (see below). MC means deciding which activities of the method should be performed and to what extent. Since a Development Track is a configuration of the base method with a focus on one specific project characteristic, it involves decisions about the performance of parts of the base method. Hence, we need a classification of how a part of the base method should be performed. Table 1 shows such a classification, which has proven useful in our work.

The classification in Table 1 is based on two dimensions. The vertical dimension illustrates how much *attention* should be devoted to a particular activity. If we find an activity unimportant, it can be classified as a 'Skip'. This was the case with the database design activities in the example above. The activity can alternatively be performed with the usual

prescribed attention (perform), or we can choose to put more effort into performing the activity than prescribed in the base method (emphasize).

| | | **Enhancement** | | |
|---|---|---|---|---|
| | | Extended | As is | Reduced |
| **Attention** | Emphasize | *Emphasize extended* | *Emphasize as is* | *Emphasize reduced* |
| | Perform | *Perform extended* | *Perform as is* | *Perform reduced* |
| | Skip | *Skip* | *Skip* | *Skip* |

**Table 1:** Performance of activities in the base method deduced from the categories of attention and enhancement.

The three aspects of the horizontal dimension: 'Extended', 'As is' and 'Reduced' cut through the vertical dimension. These aspects, referred to as required *enhancement*, tell to what extent the various steps that make up an activity should be performed or not. If we reduce the activity, some steps are abandoned. On the other hand, an extended activity means using steps not specified in the base method. These steps could be provided through integration with other methods or through explicit method development. In this case we are close to 'traditional' situational method assembly (e.g., Brinkkemper *et al*., 1999). The two dimensions result in a matrix classifying an activity into seven possible categories as shown in Table 1. In order to classify activities according to this scheme, we propose to use the analytic tool shown in Figure 1.

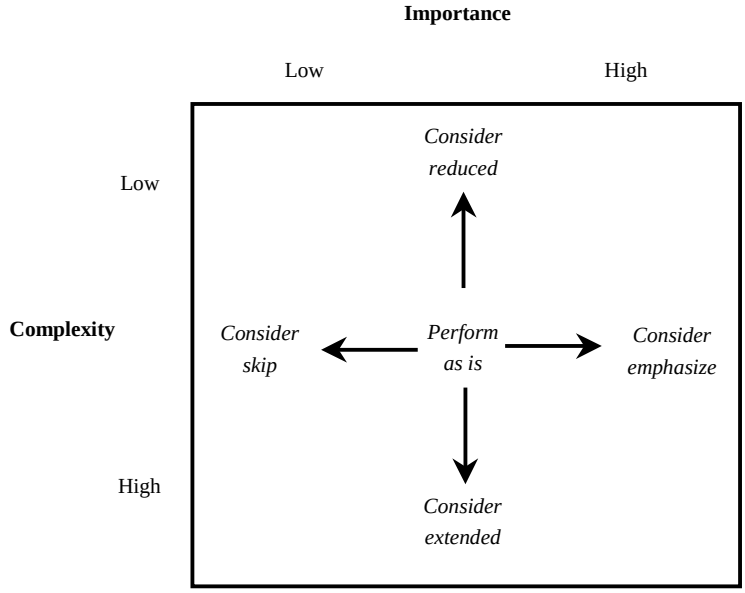


**Figure 1:** Suggested ways to perform the different activities of the base method derived from the importance and complexity of the areas with which they are concerned.

With this tool, the classification of activities is based on the combination of the importance and the complexity of the area of concern that the particular activity addresses. If the complexity is low, we may consider reducing the activity. If it is high, it might need to be extended. If it is unimportant, we may consider skipping it. Finally, if it is very important it might require emphasis. Note that the centre of the cross in Figure 1, which represents the 'normal case' of the base method, might slide within the importance and complexity space depending on what is 'normal' in the current organization; diverging from the normal situation means moving along the Importance-axis or the Complexity-axis.

Classifying an activity as 'Skip' naturally means that we do not have to decide whether to perform it 'Reduced', 'As is' or 'Extended'. A 'Skip' is always a 'Skip'. For example, an activity addressing an irrelevant area of concern for the project should not be performed. The other extreme, an area of concern that is more important than usual, might require an emphasized process. This movement can be combined with movement along the complexity axis. In the case where the area of concern was irrelevant, it can also be a complex issue to grasp. Then it could be both time and cost consuming and hence such an activity could be considered as a 'Skip'. With reduced complexity it is even more intuitive to map the activity as a 'Skip'. If the situation at hand is more complex than usual and is central to the success of the project, it is natural to consider emphasizing the activity. It should also be considered if there is a need to extend the base method or if it can be handled on an 'As is' basis.

An activity classified as 'Perform' can be carried out with a reduced, normal or extended number of steps. Thus, 'Perform as is' is the normal 'off-the-shelf' case of the base method. The 'Emphasize reduced' means reducing the number of steps, but that the remaining steps are more important than usual. Consequently, more effort will be put into the remaining steps than in the normal case. Consider, for example, an activity involving refinement of two artefacts during two separate steps, where the step concerning the first artefact should not be performed (hence a reduction of the activity), but where the performance of the second step is crucial (and hence should deserve more attention and be emphasized). This way of thinking about the Emphasize categories applies to the remaining two attention categories as well; the performed steps are more important than usual, irrespective of whether they are part of the base method or not.

### 3.2 Layer of Granularity

Until now, we have discussed 'activities' and 'steps' of the base method in a rather intuitive fashion. In this section we will be more specific by following the above-mentioned classification of method fragments into layers of granularity and perspective (as suggested by Brinkkemper *et al*., 1999). Table 2 shows the mapping of RUP concepts in a process perspective onto the 'layer of granularity' hierarchy.

| **Layer of granularity** | **RUP from a process perspective** |
|---|---|
| Method | RUP |
| Stage | Core Workflow |
| Model | Workflow Detail |
| Diagram | Activity |
| Concept | Step |

**Table 2:** Layers of granularity in RUP.

Choosing the appropriate layer of granularity to perform the analysis and, with the RUP as a base method, represent Development Tracks, is one of the most important decisions to make. This must be done with an accurate balance between precision and cost (cf., ter Hofstede and Verhoef, 1997). Choosing concept as the unit of analysis will yield high precision, but might also yield intractable amounts of work and data produced. With the other extreme[1], stage, the precision is probably too low to be useful. In our empirical work at Volvo IT, we have tried to use the model layer as the unit of analysis. However, when using RUP as the base method, it has proven difficult to present the results from MC at a Workflow Detail level. This is due to the way the RUP suggests we represent the configured process in a Development Case. It has not been possible to show what results are produced during the Workflow Detail, why they are produced, and what effects they have on other parts of the process. The latter aspect is very important to keep in mind. If a part of the process is classified as 'Skip', then it can, and very likely will, have effects on other parts of the process. The most evident case is artefacts not produced that later are presupposed as input to other

[1] Of course, Method would be the real extreme. However, whether or not to perform the method at all is another type of discussion and should have been settled before implementing the method in the organization. Otherwise it is a waste of resources.

parts of the process. In such a case there will be a domino effect, where later parts of the process are necessarily classified as 'Skip' or 'Perform reduced'. Alternatively, these later parts of the process might be considered important, implying that preceding parts can actually not be excluded.

The diagram layer (i.e., the activity level in the RUP) gives us the possibility to manage this situation with an acceptable balance between precision and cost. Each activity has a pragmatic purpose and the results are presented through the artefact list. In the RUP, artefacts used as input to an activity and the activity's resulting artefacts are visualized in artefact lists in Development Cases, showing how the activities are intertwined. However, one should be aware that by choosing this level, details on exactly how, and what parts of the results are used in following activities, are lost.

### 3.3 Representing Development Tracks

As suggested above, a development track can be understood as an array of dimension n, where n is the number of activities in the base method. Each activity is then mapped onto the set {Skip, Perform as is, Perform reduced, Perform extended, Emphasize as is, Emphasize reduced, Emphasize extended}, which represents the different ways activities can be performed. Given that the base method is carefully represented in, for example, a database, generation of descriptions of the different Development Tracks is straightforward.

### 3.4 Combining Development Tracks

Combining Development Tracks means bringing them together into a situational method. This method is then used in the specific project. The combination is based on the representation of Development Tracks and hence the result can be represented in the same way. The central issue to deal with is the outcome of a certain set of inputs. The easy cases are those where all combined Development Tracks are pointing in the same direction with regard to an activity. If every Development Track says 'Perform reduced', the activity should be performed reduced in the combined situational method. Unfortunately, the world is rarely that simple. Hence, more advanced combinations must be considered and they need to be based on empirical work. Our work indicates that it is at least possible to find heuristics for the combination of Development Tracks. To some extent it seems possible to even find a solution that can be handled algorithmically. However, at the time of writing, the MC process has been in use for such a short time that it has not yet been possible to find stable combinations. When combining Development Tracks we also need to consider how activities are interrelated. In the analysis of one single Development Track as described above, this issue occurred within the Development Track, and now the dependency is between Development Tracks. This tends to make the combination complex, and tool support could be very useful, if not essential.

### 3.5 Identifying Development Tracks

Obviously, identifying appropriate characteristics, and their corresponding Development Tracks, is an important task in order to succeed with MC based on Development Tracks. We propose this be done by a thorough analysis of the development organizations' typical types of artefacts and projects. One way of approaching this is to use an analytic framework that helps analysts to direct attention to relevant phenomena. At Volvo IT, we have used such a framework based on the concept of actability (Karlsson *et al*., 2001). During that analysis we have, at the time of writing, identified a handful of potential Development Tracks, such as 'Existing functionality with new GUI' and 'Completely new system' corresponding to the values from the dimension of the characteristic 'Degree of new functionality' as mentioned above. An important thing to bear in mind is that the collection of Development Tracks is supposed to evolve as development projects are carried out and new types of situations arise. The resulting continuous identification and refinement of Development Tracks goes hand in hand with the identification of Generic Project Types, towards which we will now turn.

## 4. GENERIC PROJECT TYPES

A *Generic Project Type* represents a set of recurrent project *characteristics*. It can be thought of as a predefined combination of Development Tracks common within the organization. The use of, and need for, Generic Project Types is a result of our empirical work with the MC process at Volvo IT. Creating and rebuilding a project specific Development Case is possible since it is derived from Development Tracks, as long as the Development Tracks it is derived from are not changed. However, it is not an efficient use of resources. It means starting at a lower configuration level than necessary, and hence more work is needed in the start-up phase of each project. Many projects are similar with respect to certain characteristics. Merely because the analysis of the characteristics is performed efficiently when it is isolated, does not mean that once it is done it cannot be represented at a more generic level.

However, more important is the fact that Generic Project Types give an opportunity to base MC on experience gathered during earlier projects. Development Tracks can be combined in a wide range of ways. There is a selection of those most interesting for a specific organization and where the projects typically have certain differences with respect to the corresponding characteristics. This selection has to be based on experience about their frequency and the differences in characteristics. As described above, combining Development Tracks cannot be fully automated. There are situations where the process engineer has to take a position on an activity and its results, decisions that cannot be represented within a Development Track. Hence, we need a container to represent this aspect in our proposal for a method configuration process. As an example of a Generic Project Type, consider the case where a legacy system should be transformed into a Web application. One of the characteristics to consider for these types of projects is the degree of new functionality that is to be developed. The Generic Project Type can be formally described as follows:

```
Generic_Project_Type Web_publishing {
Tracks:
   ...
   Degree_of_new_functionality = Existing_functionality_with_new_GUI;
   ...
 Combination:
   ...
   Activity : Analyse_Behavior {Performance = Perform_as_is;};
   Activity : Design_Components {Performance = Perform_as_is;};
   Activity : Design_the_Database {Performance = Skip;};
   Activity : Refine_the_Architecture {Performance = Perform_reduced;}
   ...
};
```

In this example of Generic Project Type we have chosen the dimension value 'Existing functionality with new GUI' as the Development Track from which to decide the performance of the base method's prescribed activities. Note that there will be a statement of chosen value (with corresponding Development Track) for each characteristic identified in the organization, replacing the ellipses (see the 'tracks' section in the example). This assumes that 'not applicable' is an implicit generic value possible for all characteristics, since there are obviously situations where some characteristics are totally irrelevant. Since the combination of Development Tracks into a Generic Project Type is, at least partly, based on heuristics, we need to document the classification of its 'combined' activities as well. We do this with the same notation as the classification of activities of a single Development Track (see the 'combination' section in the example).

## 5. A TENTATIVE METHOD CONFIGURATION PROCESS

From the structural view presented above we are now able to present a tentative method configuration process. This process can be divided into three sub-processes with regard to different frequencies in occurrence within a project. The most frequent sub-process is tuning a

Generic Project Type into a situational method. The second sub-process with regard to occurrence is combining existing Development Tracks into a new Generic Type Project based on a different selection of Development Tracks. Thus, this new Generic Type Project has a different profile if we examine the Development Tracks closely. The third and least frequent sub-process is the administration of specific Development Tracks. This last sub-process could involve creating a new Development Track in order to meet a new project characteristic that has not been considered in earlier projects. The tentative method configuration process with each sub-process is presented in Figure 2.

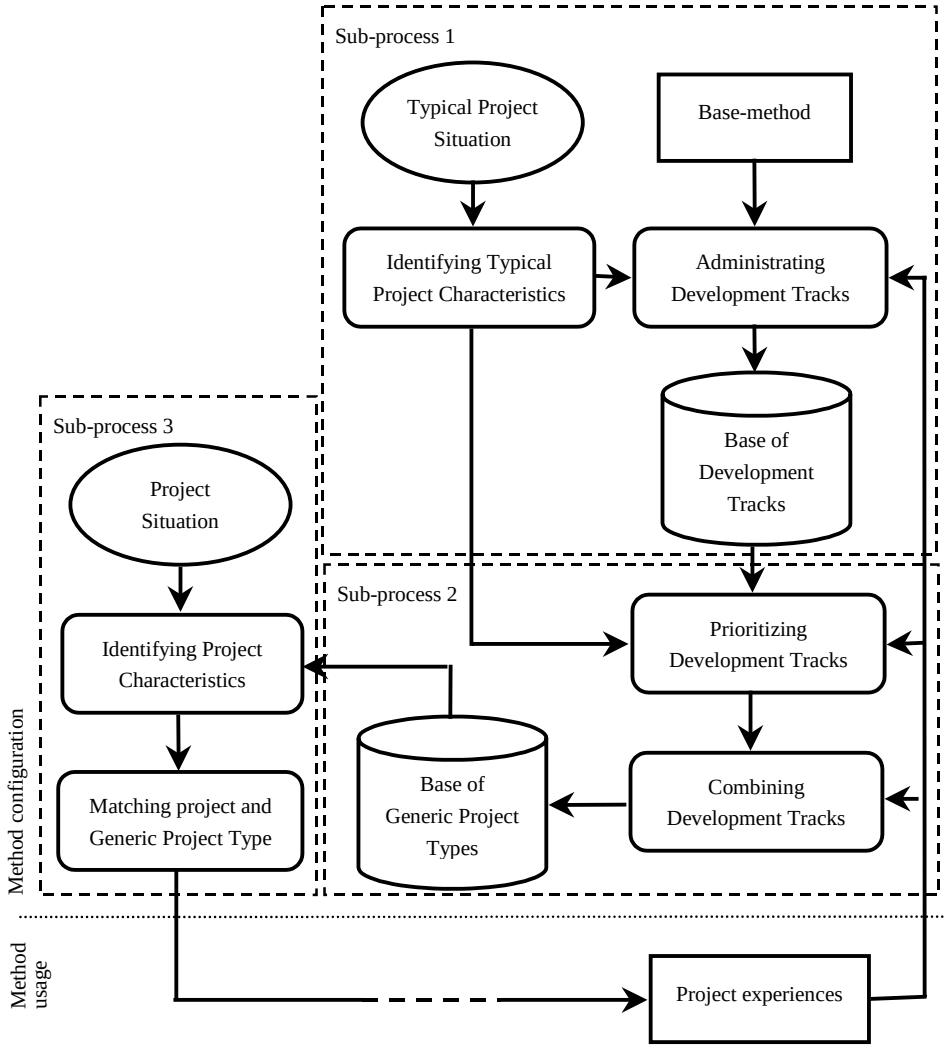


**Figure 2:** A tentative method configuration process based on Development Tracks and Generic Project Types.

Let us change focus and study the process as we would when we use it for the first time. Then we have to elaborate at least one Development Track. The starting-point, however, is deciding what method is the base method. How many Development Tracks we need to work with initially depends on which project situations we identify. From these situations we extract characteristics. They are used as input for the analysis of Development Tracks, where dimensions of each relevant characteristic determine the configuration of a Development Track. Administration of Development Tracks also involves following up projects and using these experiences for changing a Development Track. The pre-made process configurations are stored in a base of Development Tracks.

Few, if any, projects can be found with only one single characteristic that affects the total configuration of the situational method. Thus, a project will involve several Development Tracks, and so will a Generic Project Type. The sub-process of creating a Generic Project Type consists of two activities, giving priority to Development Tracks and combining them.

The selection of relevant Development Tracks for a Generic Project Type is based on characteristics for such a project and since based on the activity 'Identifying Project Type Characteristics'. The combining of Development Tracks has been briefly discussed earlier in this paper. However, an important aspect of the combination of Development Tracks into a Generic Project Type is that the process engineer can actively choose not to use a possible algorithm for combination (see above), and stick to situated heuristics. This analytical freedom can be based on not yet formalized experiences from earlier projects. Hence, it is a foundation for managing experiences and is a source for reuse, if the organization has recurrent development patterns.

Starting up a specific project means investigating the project situation and identifying project characteristics. The analysis of the project situation has to be focused in order to make it possible to classify the artefact against characteristics of Generic Project Types. The characteristics of the specific project can be represented as an array in the same manner as a Generic Project Type, in order to create a profile. If a matching Generic Project Type is found, it can be the basis for tuning it into a project specific Development Case and thus a situational implementation of the base method. The project results will include at least one artefact and some useful experience. From the method configuration process perspective, it is important to have feedback based on those experiences. Otherwise, the process becomes linear and it is not possible to improve the Development Tracks, Generic Project Type, and most important of all, future projects.

## 6. RELATED RESEARCH

Since much research within the ME field has been devoted to situational ME, taking the artefacts' characteristics and their context into consideration is not a new concept. It has been proposed before by Harmsen (1997), among others. Van Slooten and Hodes (1996) discussed a contingency model based on contingency factors. Their model is based on earlier research about such factors, and from the literature it is obvious that much effort has been put into finding these factors.

Harmsen (1997) uses these factors as input for method assembly. Project characterization is used for selection of method fragments and route maps from a method base. We can conclude that similarities exist with our MC process, and Harmsen (1997) and Odell (1996) inspired our process. However, differences can be found. Our starting-point with a specific base method is different from building each situational configuration upon method fragments for each part of the method chain. Since we have a base method as a shell, the team members can build their core competency around this base method.

Furthermore, since we use Development Tracks and Generic Project Types, we start at a more aggregated level compared to method fragments. Through accumulation of experiences, the organization will generate a more precise selection of Generic Project Types and thus manage MC more efficiently. Even in those cases where no Generic Project Type exists, the resulting effort from producing a new Development Track will finally add a new Generic Project Type. A new Generic Project Type does not have to involve a new Development Track, just another selection among them. Thus, with recurrent development patterns in an organization, the configuration process should be more efficient. The use of Generic Project Types also yields a better foundation for managing project experience.

Brinkkemper (2000) presents the Baan Development Method, by using concepts which are similar to ours. However, in his work the focus is on online presentation of and access to methods rather than MC. This Web-enabling of methods is of course central to our work since RUP in itself is described in HTML, which is a Volvo IT requirement on our approach as well.

## 7. CONCLUSION

Method Configuration (MC) has in this paper been treated as a specific kind of Method Engineering (ME). The main characteristic of MC is that it uses one specific method as a basis for creating project specific configurations—a *base method*. In this paper we have

introduced the concepts of *Development Track* and *Generic Project Types*, which aim to ease the burden of configuring the base method given certain project characteristics. A Development Track has been described as a pre-made process configuration designed to fit a specific development characteristic. Furthermore, Development Tracks can be packaged into Generic Project Types representing recurring development patterns in the particular development organization. An example of a Development Track could be 'Existing functionality with new GUI' which would correspond to one possible *value* of a *characteristic* 'Degree of new functionality'. This characteristic and corresponding Development Track could furthermore be part of a Generic Project Type 'Web Publishing'.

The benefits of using these concepts are that MC can be performed more efficiently since pre-made configurations can be used over and over again. Hence, there is no need to perform a complete method assembly for each new project. Furthermore, it seems that experiences can be gathered and reused more efficiently since they can be attributed to coherent sets of activities common in the organization, rather than to context-free atomic actions.

The approach with Development Tracks and Generic Project needs further empirical work. As a tentative process more research has to be put into making it stable. There is a need to address the issue of how to extend or exchange larger parts of the base method. In the tentative process we have the possibility to classify existing activities as 'skip', but we also need to be able to add activities. In doing so we could then handle larger building blocks such as exchange of workflows in RUP, since it consists of activities; that is, allowing for discussions about enhancement and attention at different layers of granularity. However, the process is the foundation for future work and it will be used as a tool for process configuration. The follow-up to that work will be evaluating the consequences from the process configuration, thus grounding the process on an action and consequence level (Goldkuhl, 1999). In parallel to this empirical grounding, more effort has to be put into theoretical grounding of the approach as well, i.e., into verifying its consistency internally and in relation to other ME approaches.

Another important future research topic is to investigate the design of a computerized tool to support the proposed MC approach. At Volvo IT, such a tool probably needs to be integrated with the Rational Process Workbench, which is a tool for configuration of the RUP. From our research point-of-view, however, it is important to maintain method independence and allow for different base methods. Hence, one obvious future research topic is to apply the proposed MC approach to other base methods in other organizational contexts. Along with the work on tool support, the (highly preliminary) metamodelling language used to describe Development Tracks, Characteristics, and Generic Project Types in this paper needs to be carefully redesigned, probably with inspiration from existing method engineering languages (e.g., Harmsen, 1997). For example, the language must be able to appropriately describe configuration rationale, such as decisions made about classification of activities and combination heuristics.

## ACKNOWLEDGEMENTS

This work has been financially supported by the Swedish Knowledge Foundation. We would also like to thank Boris Karlsson and Maria Hagström at Volvo IT, and Per Löfnertz, consultant from Rational engaged by Volvo IT, for participating in the work that has led to the proposed approach to method configuration discussed in this paper.

## REFERENCES

Ågerfalk P J, Åhlgren K (1999). Modelling the Rationale of Methods. *Managing Information Technology Resources in the Next Millennium*, pp. 184–190, Khosrowpour M, ed., Proceedings of the 1999 Information Resources Management Association International Conference. Hersey, PA, USA, May 16–19, 1999.

Beck K (2000). *Extreme Programming Explained: Embrace Change*. Addison Wesley Longman, Inc. Reading, MA, USA.

Brinkkemper S (1996). Method engineering: engineering of information systems development methods and tools. *Information and Software Technology* 38, pp. 275–280.

Brinkkemper S, Saeki M, Harmsen F (1999). Meta-modelling based techniques for situational method engineering. *Information Systems*, vol. 24, no. 3, pp. 209–228.

Brinkkemper S (2000). Method Engineering with Web-enabled Methods. In *Information Systems Engineering, State of The Art and Research Themes*, Arne Solveberg (ed). Springer-Verlag, London, Great Britain.

Checkland P (1981). *Systems Thinking, Systems Practice*. Wiley, Chichester, UK.

Cronholm S, Ågerfalk P J (1999). On the Concept of Method in Information Systems Development. *Proceedings of the 22nd Information Systems Research Seminar in Scandinavia* (IRIS 22), Keuruu, Finland, August 7–10, 1999.

Goldkuhl G (1999). *The Grounding of Usable Knowledge: An Inquiry in the Epistemology of Action Knowledge*. CMTO Research Reports, Linköping University, Sweden.

Goldkuhl G, Lind M, Seigerroth U (1998) Method integration: The need for a learning perspective. In: Magee J. N., Sommerville I., Jayaratna N. (Eds.) *IEE Proceedings Software: special issue on Information Systems Methodologies*, **145**, (4), pp. 113–118.

Harmsen A F (1997). *Situational Method Engineering*. PhD Dissertation. Moret Ernst & Young Management Consultants, Utrecht, Netherlands.

Hofstede A H M, ter Verhoef T F (1997). On the Feasibility of Situational Method Engineering. *Information Systems*, Vol. 22, No. 6/7, pp. 401–422.

Karlsson F, Ågerfalk P J, Hjalmarsson A (2001). *Demystifying the Internet-based Software Artefact*. To appear in Proceedings of the 10th International Conference Information Systems Development (ISD2001), 5–7 September 2001, Royal Holloway, UK.

Kruchten P (1999*). The Rational Unified Process: An Introduction*. Addison-Wesley, Reading, MA, USA.

Odell J J (1996). A primer to method engineering. *Method Engineering: Principles of method construction and tool support*, Proceedings of the IFIP TC8, WG8.7/8.2 Working conference on method engineering, 26–28 August 1996, Atlanta, USA.

Rolland C, Prakash N, Benjamen A (1999). A Multi-Model View of Process Modelling. *Requirements Engineering*, (4), pp. 169–187.

Rumbaugh J (1995). *What is a method?* Technical Paper, Rational Software, Inc.

Slooten K van, Hodes B (1996). Characterizing IS development projects. In *Method Engineering: Principles of method construction and tool support*, Proceedings of the IFIP TC8, WG8.7/8.2 Working conference on method engineering, 26–28 August 1996, Atlanta, USA.