# Analyzing a Software Process Model Repository for Understanding Model Evolution

Martín Soto
Alexis Ocampo
Jürgen Münch

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany
{soto, ocampo, muench}@iese.fraunhofer.de

**Abstract:** Process models play a central role in the process improvement cycle. Often, large process models evolve in an ad-hoc manner, a fact that may easily have critical implications such as increased maintenance effort. This highlights the need for supporting the control and management of process model evolution, a kind of support that is currently widely missing. Analyzing existing model repositories in order to better understand model evolution can be seen as a first step towards identifying requirements for process model evolution support. This article presents a study that analyzes the evolution history of a large process model with the purpose of understanding model changes and their consequences. Besides the study description, the article provides an overview of related work, and suggests open questions for future work.

**Keywords:** process modeling, process model change, process model evolution, model comparison, V-Modell XT, Evolyzer

## 1 Introduction

Process models play a central role in the process improvement cycle. On the one hand, process analysis activities intended to identify improvement opportunities use models as one of their main inputs. On the other hand, the process model (usually in the form of a process guide) constitutes the main support process actors have in order to enact the process accurately. For this reason, any proposed process improvements will only be enacted if they are added to the model first. The consequence is that process models must be maintained in lockstep with the process itself, in order for controlled process improvement to happen in a sustained fashion.

Given its importance for process improvement, as well as the large size and complexity of many industrial process models, it comes as a surprise that support for managing model evolution is still widely missing. Standard version management tools are generally barely adequate for the task of storing a model's version history, or observing and analyzing the changes that have happened to it. This is in stark contrast to the situation in code evolution, where version management has been practiced for decades, and countless research efforts have been devoted to analyzing the resulting version histories.

It it this lack of proper evolution support for process models that motivated us to perform the empirical study presented in this article. In the study, which is a follow-up to previously published work by the authors [1], we analyzed a set of 604 development versions of a large process model with the purpose of identifying change patterns and understanding their effect over time. The underlying assumption is that, given the size and complexity of the studied model, its development process will have a behavior similar to that of a standard software development process over time. In order to perform the study, we used our *Evolyzer* model comparison system to compare versions along the history pairwise, and produced a database of detailed changes that was, in turn, the subject of graphical and statistical analysis.

We see the main contributions of this work at two different levels. At the level of the concrete study, our results provide some further evidence of the similarity between the evolution of large models and that of industrial software systems. These similarities suggest, in turn, that in order for model-based development to succeed, support for model evolution must be improved until it is at least as good as support for code evolution is now. Furthermore, at a more general level, our study can be seen as a demonstration of the practical feasibility of observing and analyzing the evolution of complex process models, as well as of the suitability of our model comparison tools for this purpose. In this sense, we expect our work to provide a basis for more advanced empirical work on process model evolution in the future.

The rest of the paper is organized as follows: Section 2 describes the execution of the empirical study and analyzes its results, Section 3 briefly surveys related research work, and Section 4 presents the main conclusions of the work and discusses a number of research questions resulting from the present work that we would like to address in the future.

## 2 An Empirical Study on Model Follow-up Change

As stated above, the assumption underlying our empirical work on process model evolution is that there exists a strong parallel between model evolution and general software evolution. For this study, we concentrated on one particular aspect, namely, the effect of changes on stability. Our central research question was whether changes made to a model are likely to introduce problems that must be corrected in follow-up changes. A positive answer to this general question would imply that, similar to the case of code development, projects dealing with the development and maintenance of complex models have to plan for a stabilization period when doing extensive changes.

### 2.1 The German V-Modell XT

We investigated this main research question in the context of a large software process model, the German V-Modell® XT. The V-Modell XT [2] (not to be confused with Royce's V-Model [3]) is a high-level process description that is currently being adopted as the software development standard for the German public administration. It covers such aspects of software development as project management, configuration management, software system development, and change management, among others. In printed form, the latest English version at the time of this writing (version 1.2.1) is 765 pages long and describes about 1500 different process entities. Internally, the V-

Modell XT is structured as a hierarchy of process entities interconnected by a complex graph of relationships. This structure is completely formalized, and suitable for automated processing. The actual text of the model is attached to the formalized structure, mainly in the form of entity and relationship descriptions, although a number of documentation items (including a tutorial introduction to the model) are also integrated into the structure in the form of so-called text module entities.

Actual editing of the model is performed with a software tool set created specifically for this purpose. The printed form of the V-Modell XT (a process guide) is generated automatically by traversing the structure in a predefined order and extracting the text from the entities found along the way. The V-Modell XT contents are maintained by a multidisciplinary team of experts, who work, often concurrently, on various parts of the model. In order to provide some measure of support to this collaborative work, the model is stored as a single XML file in a standard code versioning system (CVS). As changes are made by the team members, new versions are created in this system. Being a standard versioning system intended for code, CVS is able to store a version history and maintain a simple change log, but can hardly provide useful information about the actual changes done in each version. In particular, the output of the *diff* program used by CVS to compare versions cannot easily tell which entities were affected by a version or in which way they were changed.

The change logs show that, since its initial inception, the model has been changed often and for a wide variety of reasons. Changes may be as simple as individual spelling or grammar corrections, or as complex as the introduction of a whole set of processes for hardware development and software/hardware integration. The richness and complexity of this change history makes the V-Modell XT a very interesting target for evolution analysis.

The descriptive analysis we performed in our exploratory study [1] showed that much of the changing activity concentrated around public releases of the model and affected some process modules much more than others. While looking at the changes that happened to model entities, both at the aggregated process module level and at the detailed single entity level, one phenomenon was apparent in the graphs, namely, that "bursts" of activity could be observed that tended to calm down after a few versions. The present study concentrates on these bursts, with the aim of determining if they constitute a significant evolution pattern for the V-Modell XT.

### 2.2 Hypotheses

One possible way to explain the activity bursts (and probably one that would be rather obvious to anyone familiar with software development) is that *primary changes*, that is, changes intended to introduce new features or to restructure the model, often introduce defects that have to be corrected later on, by doing a number of secondary or *follow-up* changes. So, one activity burst would actually consist of a primary change, maybe split into a few versions, and a number of follow-up changes intended to reestablish model correctness. Proving this conjecture is difficult, however, since it would require an objective classification of changes into primary and secondary ones, a task that would most probably require human judgment in many cases.

Still, we can target a weaker form of the conjecture, namely, that changing an area of the model increases the probability of changes happening to the same area in the near future. This would mean that activity bursts observed by visual inspection of the

graphs have statistical significance, that is, they cannot be simply explained by chance, or by artifacts of the graphical representation used.

One difficulty that arises here is that of defining what exactly an "area" of the model is. Actually, given the complex structure of the V-Modell XT, there would be a number of potential, reasonable definitions, covering various levels of granularity. As an initial step, we decided to work at a fine level of granularity, and analyze changes at the entity level. The main rationale for this decision is that if we can observe the phenomenon at the entity level, it holds also at least for the larger entity containers, whereas the opposite cannot be stated.

The previous considerations led us to the following two hypotheses:

H1: Changing a process model entity in a particular version increases the probability of changing it again in subsequent versions.

H2: Changing a process model entity at a given date increases the probability of changing it again in the following days.

In the hypotheses, we are not making any statements about the particular way in which the probability of further changes should increase after a change. Our current knowledge of the evolution of this and other models is still too limited to provide a more detailed mathematical model of how a change affects the probability of future changes to the same area.

One conclusion that would immediately follow from the hypotheses above is that changes to an entity in the various versions in the history are not independent events: that is, changes to an entity affect the likelihood of future changes to the same entity. Based on this observation, we formulate our null hypotheses as follows:

$H1_0$: Changes to a process model entity in a particular version are independent from changes to the same entity in all other versions. Moreover, there is a fixed probability $p_v$ of changes occurring to an entity in a particular version, for all versions and for all entities present in each version.

$H2_0$: Changes to a process model entity on a particular day are independent from other changes to the same entity. Moreover, there is a fixed probability $p_d$ of changes occurring to an entity on a particular day, for all days in the studied period and for all entities present in the model during that period.

Notice that, if falsified, these null hypotheses are weaker than the negation of the alternative hypotheses, namely, they would show that there exists a dependency between changes in different versions and at different points in time, but they would not guarantee that the probability of follow-up changes actually increases. We will address this point later.

### 2.3 Data Preparation

As for the initial study, the first step we took in order to make it possible to analyze the V-Modell's change history statistically was to read a sizable portion of the V-Modell XT's versioning history into our *Evolyzer* model comparison system. Although a description of the internal operation of *Evolyzer* is beyond the scope of this paper (see [4] for details), a short explanation of its workings is in order. The basis of the system is a model database that can contain an arbitrary number of versions of a model. Model versions in the database are represented using the RDF notation [5], and the whole model database can be queried using a subset of the SPARQL [6] query language for RDF.

The main purpose of *Evolyzer* is to allow for comparing model versions from the database. Given two arbitrary versions, the system computes a so-called *comparison model,* which contains all model elements (RDF statements, actually) present in the compared versions, marked with labels indicating whether they are common to both versions or are only present in one of them and, in the latter case, which of the versions they come from. Given the high level of granularity of this comparison, identifying changes in it by direct inspection is generally a difficult task. For this reason, change identification is performed by defining special *change patterns* (see [4] for a detailed explanation) that match particular types of changes in the comparison model. *Evolyzer* provides an efficient interpreter for the pattern language, which can identify instances of a particular pattern in the comparison model of two arbitrary versions.

For the present study, we attempted to read 604 versions from the original versioning repository into our system. These versions were created in a little more than two years' time, with three major and one minor public releases happening during that period. Since *Evolyzer* uses the RDF notation for model representation (this is necessary in order for our comparison technique to work at all), each V-Modell version was mechanically converted from its original XML representation into an RDF model before reading it into the system. This conversion did not add or remove information, nor did it change the level of formalization of the original process description. The conversion process was successful for all but 4 of the 604 analyzed versions. These four versions could not be read into our repository because their corresponding XML files contained syntax errors, and they were replaced by copies of the previous version in order to prevent our system from reporting spurious changes.

After importing the version history, we proceeded to compare the versions pairwise to identify individual changes happening from one version to the next. As changes, we considered the addition or deletion of entities, the addition or deletion of relations between entities, and the alteration of text properties. We identified these changes by defining corresponding change patterns and searching for them in the version comparisons. Information about each of the identified changes, including type, version number, and affected process entities, was encoded in RDF and stored in the repository together with the model versions. This allowed us to easily go from the change information to the actual model contents and back from the models to the changes as necessary for our analysis (see [7] for the details of how this cross-referencing works).

### 2.4 Data Analysis and Interpretation

In order to test our first hypothesis, we proceeded to look for *pairs of consecutive changes* in the model history. A pair of consecutive changes is defined as a triple *(e,* $v_1, v_2$*)*, with *e* a model entity, and $v_1$, $v_2$ version numbers, such that

*1.* $v_1 < v_2$.

*2.* $v_1$ and $v_2$ contain changes that affect *e*.

3. no version *v,* with $v_1 < v < v_2$, contains changes affecting *e*.

In other words, these are pairs of versions that change the same entity, with none of the versions lying between them affecting the entity.

If our first null hypothesis $H1_0$ holds, the probability of an entity being changed by a particular version has a fixed value $p_v$. This, in turn, implies that the process of changing an entity over its history can be modeled as a Bernoulli process with proba-

bility $p_v$, where each new version containing the entity is seen as a Bernoulli trial and the trial succeeds if the corresponding version actually changes the entity.

Let us now consider the length $v_2 - v_1$ of a pair of consecutive changes. If the change process is actually a Bernoulli process, the probability $P(l)$ of the pair having a particular length $l$ would be given by the geometric distribution, that is

$$P(l) = (1 - p_v)^{l-1} \cdot p_v$$

This formula is easily understood as the probability of making $l - 1$ unsuccessful trials followed by one final, successful trial.

In order to determine if this is actually the case in the V-Modell history, we queried our model evolution database to find all text changes (changes to text attributes) affecting entities in the model during the observed period, and, with the help of some simple postprocessing of the query results, identified all pairs of consecutive changes in the history as defined above. We found 2835 individual pairs during the period studied.

Figure 1 is a histogram of the lengths of these pairs, with categories of width 10. 47.3% of the pairs have a length of 10 versions or less, 71% of 50 versions or less, and 90.2% of 170 or less.

As explained above, if the null hypothesis holds, this observed distribution should correspond to the geometric distribution for the probability $p_v$. In order for a goodness-of-fit test to be possible, it is necessary to estimate a reasonable value for $p_v$. We tried two different methods for estimating this value.

The first method is based directly on the null hypothesis. If the probability of making changes to any particular entity in any particular version is always the same, we can look at the complete history as a single Bernoulli process in which the individual histories of the various process entities are placed in a single row in some arbitrary order. The total number of trials in that process would then correspond to the sum of the lengths of the individual histories of the entities in the model. Since entities are introduced and deleted along the history, the length history varies from one entity to the next. Using database queries for the creation and deletion points of entities, we calculated the total number of trials to be approximately 1.150.000 and the total number of changes to be 4248, producing a value of 0.0037 for $p_v$. The curve for the resulting geometric distribution is compared with the original histogram in Figure 1, where it is

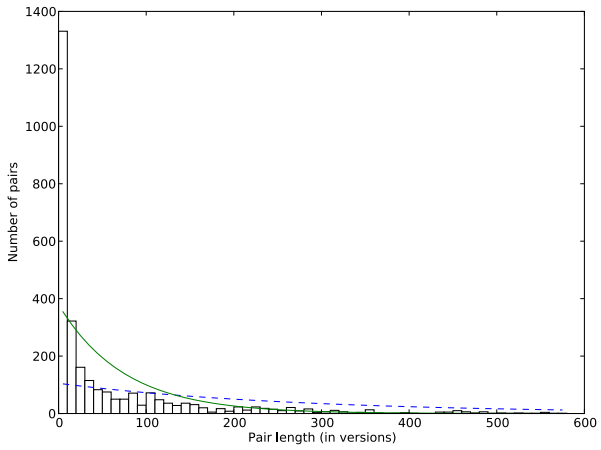


**Fig 1:** Distance in versions for consecutive entity changes.

shown as a dashed line. A chi-square goodness-of-fit test for this case yielded a p-value smaller than 0.0001.

The second method we tried in order to determine the value of $p_v$ was to use a standard optimization procedure to find a value of $p_v$ that minimizes the chi-square value with respect to the actual data. The $p_v$ value obtained was 0.0133, with a p-value for the goodness-of-fit that is still below 0.0001. The curve for this probability value is shown in Figure 1 as a solid line. Given the very low probability obtained in both cases for the chi-square tests, we can reasonably reject our first null hypothesis, $H1_0$.

The direct implication of rejecting the null hypothesis is that we are observing a certain level of dependency among changes. Still, it is not clear if this dependency really implies a *higher* probability of subsequent changes after a change. The comparison in Figure 1, however, shows that the first categories are much higher than those predicted by the estimated geometric distributions. This means that there is a high number of short pairs (indicating changes that are very close to each other) that could not be explained if changes were assumed to happen with a fixed probability. This supports our alternative hypothesis H1.

For the second hypothesis, we extended the previous analysis to also consider the time when changes were made. For each of the identified pairs of consecutive changes, we measured the distance in days between the check-in operations corresponding to the versions $v_1$ and $v_2$ in the pair, and discarded those pairs where the changes happened on the same day. This left us with 2324 of the original 2835. Figure 2 contains a histogram of the distances obtained, with the categories corresponding to 10-day intervals.

The first approach used in the previous case to estimate the value of $p_v$ cannot be used here as easily, because the number of entities in the model can vary in the course of a single day. For this reason, we used only the chi-square optimization method to estimate the value of the probability $p_d$ of an entity being changed (at least once) on a particular day. The resulting curve can be seen as a solid line in Figure 2. The actual resulting value was 0.0133 with a goodness-of-fit p-value also below 0.0001. The conclusion is analog to the one for the previous case: The null hypothesis $H2_0$ can also be rejected in this case. Similarly, the pronounced peak in the first category observed

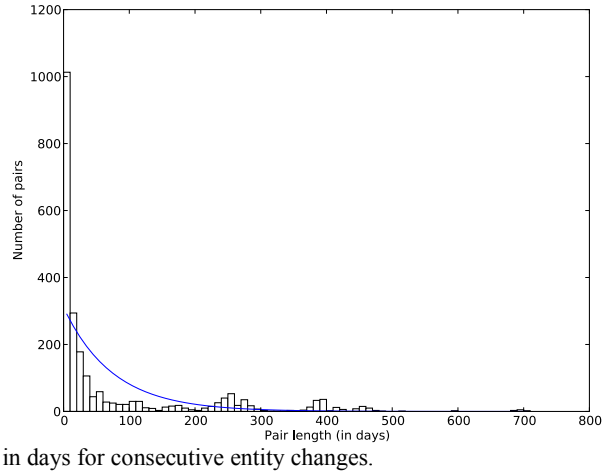


**Fig 2:** Distance in days for consecutive entity changes.

in the graph over the theoretical distribution contributes the remaining support to our alternative hypothesis H2.

### 2.5 Threats to Validity and Limitations

Although the high significance of the previous results clearly shows that the null hypotheses can be rejected, at least in the stated form, the question remains of whether the assumption of a constant change probability over all entities in the model, and for all model versions (or days in the studied period) actually holds in practice. For the time case, for instance, activity increases around releases, which would lead to shorter consecutive change distances in days for the time periods around releases. This effect, however, would be only observable when measuring change distances in days, but not when measuring them in versions, so the fact that the effect is observed in both of them actually speaks against this risk.

It is also quite possible that certain entity types may have higher change probabilities, and this may explain at least some instances of the short change distances we are

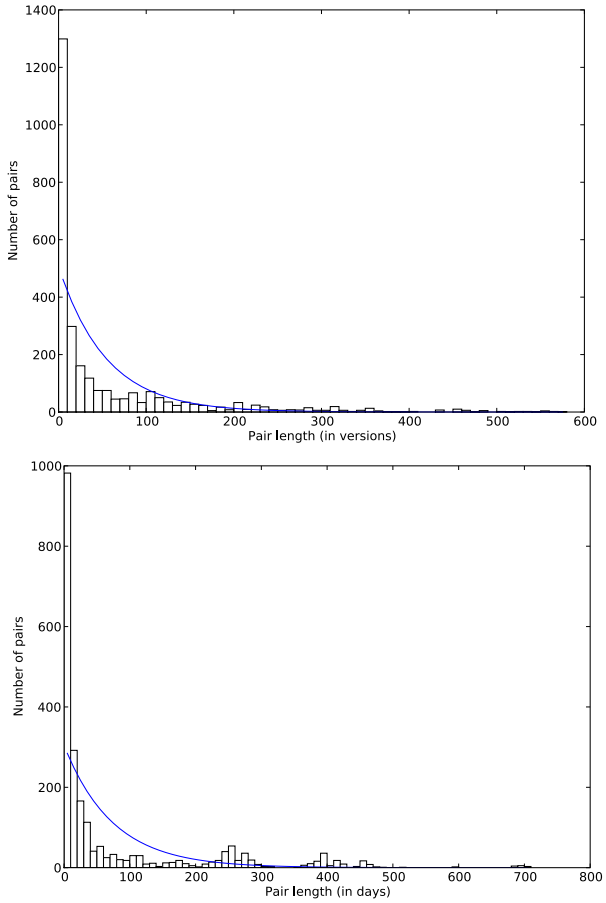


**Fig 4**: Distance in versions and days for consecutive text attribute changes.

observing. In particular, certain entity types contain more text attributes, or tend to have longer text attribute contents, thus increasing the probability of changes to them. One option for looking into this in more detail is to identify consecutive changes to the individual attribute instances. If change distances are still short for this case, we can more safely assert that the effect observed at the entity level is not only explained through differences among entity types.

Figure 3 presents the version and time histograms for pairs of consecutive changes to individual text attributes in the model. These pairs are defined in a similar way as for the entity case, with the exception of versions in the pair having to affect both the same entity and the same attribute in the entity. 2749 pairs were found for the version-based analysis, and 2225 for the time-based analysis. The best-fit geometric distributions have probabilities of 0.0182 and 0.0134 respectively. Both of them yielded p-values for the chi-square test below 0.0001. These results are consistent with our analysis for the entity case, namely, that changes to single attributes also increase the probability of future changes to the same attribute.

A larger, much more difficult question is related to external validity. Since model evolution is just starting to be studied, it is premature to say that the results observed for the V-Modell XT can be generalized to other similar models. However, one can assume that the results would apply, at least to some extent, to models such as the Rational Unified Process (RUP) [8], which have a similar purpose and level of complexity. Further studies in this direction would be very valuable.

## 3 Related Work

Much support and several studies have been dedicated to understanding software evolution. Many examples of such work can be found in recent workshops and conferences [9-11]. Most of these studies have concentrated on confirming Lehman's [12] and Parnas' [13] findings, by examining successive source code releases, or examining change data stored in source code control systems. Such studies are frequently performed with the support of advanced data mining techniques [14, 15], as is the case for both the product and process communities. In the product community, studies are performed for purposes such as understanding the evolution of programs and the programs themselves [16], detecting evolutionary coupling between files [17] and model elements (e.g., classes) [18], suggesting and predicting likely changes, preventing errors due to incomplete changes, or detecting coupling undetectable by program analysis [19].

Jazayeri [20] stated that "Individual software products age while our understanding of them and, as a result, their models (and meta-models) evolve", and encouraged the community to move the focus of studies towards the evolution of models and meta-models. As mentioned in the introduction of this paper, we currently observe a lack of empirical studies on the evolution of large models. In addition to the preliminary study that gave rise to the present work [1], two of the authors performed an exploratory study [21] with the goal of understanding the nature of process model changes in the context of the aerospace industry. That study presented the most important issues that motivated process engineers to change an aerospace software process standard.

## 4 Conclusions and Outlook

The empirical study presented in this paper had the purpose of determining whether changes to entities in the German V-Modell XT software process standard increased the probability of subsequent changes to the same entities, both in time and in the version sequence. The basic conjecture giving rise to this research question is that certain changes to the model introduce defects that have to be corrected in a number of follow-up changes, thus producing "bursts" of activity that are observable in the model history.

The data used for the study corresponded to a set of 604 consecutive versions, containing changes performed to the V-Modell XT from October 2004 to October 2006. The differences from one version to the next were calculated automatically by means of the focused identification of changes supported by our *Evolyzer* tool. The detailed description of this change identification process, as well as the definition of the change types used here, can be found in [4] and [7], respectively.

The descriptive analyses and the statistical tests presented in this article confirm the hypothesis that changes to an entity increase the probability of further changes in the future. Although this does not prove that our underlying model of primary and follow-up changes holds, it is a first step towards providing evidence in this direction.

In this sense, also, the results of the present study support our assumption that there are clear behavior similarities between code and model evolution (see [22] for an example of a study that yields similar results for software systems). This assumption has a number of consequences. On the one hand, it suggests that existing mechanisms that facilitate code maintenance can be potentially applied to model maintenance. For instance, encapsulation mechanisms help to isolate stable code parts from constantly changing parts and, therefore, help to gain better control of maintenance activities. In the case of models, the same encapsulation can be applied by (re-)structuring them in such a way that the contents of stable model entities are reused without change. There are also implications for project planning, since, as in the case of code, complex changes seem to be very likely to "destabilize" the model by introducing defects of various types. This would mean that project managers must plan for a stabilization period after introducing complex changes.

We see model evolution studies as valuable input for better support of model maintenance in the future. Particularly in the area of version management, which is clearly related to our work on model comparison, we believe that models can benefit to a large extent from existing code version management techniques and tools. However, the implementation of such techniques for models presents a number of theoretical and technical challenges that are not present in standard code version management. Our techniques for focused change identification, and the *Evolyzer* tool that realizes them, constitute a new proposal for this kind of support. The viability of this proposal can be seen in the fact that these techniques and tools provided us with the capabilities to perform the study presented in this paper.

The similarities observed between model and code evolution are also a motivation for performing future studies based on research questions already posed in the latter area. For example, in a recent code evolution study [19], Zimmermann et al. attempted to find hidden dependencies in a large software system by looking for pairs of program elements that have a strong tendency to be changed simultaneously, e.g., when one of them is changed in a given version, there is a high probability that the other one is also changed in the same version. In our opinion, a similar study could be

viably extended to models, with the goal of discovering hidden evolution connections between entities that could not be found using the techniques that we have applied so far.

## 5 Acknowledgments

We especially thank our colleagues Marcus Ciolkowski and Jens Heidrich from Fraunhofer IESE, who provided many valuable ideas and enriching discussions during the preparation of this paper. We would also like to thank Sonnhild Namingha, also from Fraunhofer IESE, for proofreading the paper.

During our work with the V-Modell XT, we had support from several members of the V-Modell development team. We would particularly like to mention Professor Andreas Rausch, Christian Bartelt, Michael Deynet, and Thomas Ternité, all currently at the Technical University of Clausthal, Germany.

This work was supported in part by SoftDiff, a project financed by the Fraunhofer Challenge Program. This work was also partially supported by the *Stiftung Rheinland-Pfalz für Innovation* through the Q-VISIT project (*Qualitätsorientierte Visuelle Software Inspektion*).

## 6 References

1. Soto, M., Ocampo, A.; Münch, J: The Secret Life of a Process Description: A Look into the Evolution of a Large Process Model In: Wang, Qing (Ed.); Pfahl, Dietmar (Ed.) ; Raffo, David Mitchell (Ed.): Making Globally Distributed Software Development a Success Story. International Conference on Software Process, ICSP 2008 - Proceedings. Berlin: Springer-Verlag, 2008, 257-268.
2. V-Modell® XT. Available at http://www.v-modell.iabg.de/ (last checked 2007-12-20).
3. Royce, W. W.: Managing the development of large software systems: concepts and techniques. In: Proceedings of the 9th International Conference on Software Engineering (1987), IEEE Computer Society.
4. Soto, M., Münch, J.: Focused Identification of Process Model Changes. In: Proceedings of the International Conference on Software Process (ICSP 2007), Minneapolis, MN, USA, May 19-20, 2007. Springer-Verlag (2007).
5. Manola, F., Miller, E. (eds.): RDF Primer. W3C Recommendation, available at http://www.w3.org/TR/rdf-primer/ (2004) (last checked 2007-12-20).
6. Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF. W3C Working Draft, available at http://www.w3.org/TR/rdf-sparql-query/ (2006) (last checked 2006-10-22).
7. Ocampo, A., Soto, M.: Connecting the Rationale for Changes to the Evolution of a Process. In: Proceedings of the 8th International Conference on Product Focused Software Development and Process Improvement (PROFES 2007) Riga, Latvia, July 2-4, 2007. Springer-Verlag (2007).
8. RUP. Rationale Unified Process. Available at http://www-306.ibm.com/software/awdtools/rup/ (last checked 2008-08-06).
9. Di Penta, M., Lanza, M. Ninth international workshop on Principles of software evolution. (IWPSE) 2007. ISBN:978-1-59593-722-3.
10. 8th International Workshop on Principles of Software Evolution (IWPSE 2005), 5-7 September 2005, Lisbon, Portugal. IEEE Computer Society 2005, ISBN 0-7695-2349-8.

11. 7th international Workshop Principles of Software Evolution (September 06 - 07, 2004). IWPSE. IEEE Computer Society, Washington, DC, .08-viii. DOI= http://dx.doi.org/10.1109/IWPSE.2004.15
12. Lehman, M. M., Belady, L. A. Eds.: Program Evolution: Processes of Software Change. Academic Press Professional, Inc, 1985.
13. Parnas, D. L.: Software Aging. Proceedings of the 16th International Conference on Software Engineering (ICSE 1994), pp. 279 – 287, Sorrento, Italy, 1994.
14. van der Aalst, W. W. T., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. Knowledge and Data Engineering, IEEE Transactions 2004. 16(9): p. 1128 - 1142.
15. Eiben, A. E., Smith, J. E.: Introduction to Evolutionary Computing. Natural Computing. Springer-Verlag, Berlin, 2003.
16. Ball, T., Kim, J. M., Porter, A. A., Siy, H. P, If Your Version Control System Could Talk: Proc. ICSE Workshop Process Modelling and Empirical Studies of Software Eng., 1997.
17. Gall, H., Hajek, K., Jazayeri, M.: Detection of Logical Coupling Based on Product Release History: Proc. Int'l Conf. Software Maintenance (ICSM '98), pp. 190-198, Nov. 1998.
18. Bieman, J. M., Andrews, A. A., Yang, H. J.: Understanding Change-Proneness In OO Software through Visualization: Proc. 11th Int'l Workshop Program Comprehension, pp. 44-53, May 2003.
19. Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S.: Mining version histories to guide software changes. IEEE Transactions on Software Engineering, volume 31, issue 6, June 2005 Page(s):429 – 445. Digital Object Identifier 10.1109/TSE.2005.72
20. Jazayeri, M.: Species evolve, individuals age Invited Keynote Talk. 8th International Workshop on Principles of Software Evolution (IWPSE 2005), 5-7 September 2005, Lisbon, Portugal. IEEE Computer Society 2005, ISBN 0-7695-2349-8
21. Ocampo, A., Münch, J.: Process Evolution Supported by Rationale: An Empirical Investigation of Process Changes. In: Wang, Qing (Ed.); Pfahl, Dietmar (Ed.) ; Raffo, David Mitchell (Ed.) ; Wernick, Paul (Ed.): Software Process Change : International Software Process Workshop and International Workshop on Software Process Simulation and Modeling, SPW/ProSim 2006 - Proceedings. Berlin: Springer-Verlag, 2006, 334-341.
22. Burd, E., Munro, M.: Evaluating the evolution of a C application. In proceedings International Workshop on Principles of Software Evolution. Available at: http://dontaku.c-sce.kyushu.ac.jp/IWPSE99/Proceedings