Using Metrics for Assessing the Quality of ASF+SDF Model Transformations*

Marcel F. van Amstel[1], Christian F.J. Lange[2], and Mark G.J. van den Brand[1]

[1] Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
{M.F.v.Amstel,M.G.J.v.d.Brand}@tue.nl
[2] Federal Office for Information Technology, Cologne, Germany
mail@christian-lange.com

Abstract. Model transformations are an essential part of Model Driven Engineering and are in many ways similar to traditional software artifacts. Therefore it is necessary to define and evaluate the quality of model transformations. We propose a set of six quality attributes to evaluate the quality of model transformations. We define 27 metrics for ASF+SDF model transformations to predict the quality attributes we propose. Metrics data has been collected from six heterogeneous model transformations automatically. The quality of the same transformations has been evaluated manually by several ASF+SDF experts. We assess whether the automatically collected metrics are appropriate predictors for the quality attributes by correlating the metrics data with the expert data. Based on the measurement results, we identify a set of predicting metrics for each of the quality attributes for model transformations.

1 Introduction

Model Driven Engineering [1] (MDE) is a software engineering discipline in which models play a central role throughout the entire development process. MDE combines domain-specific modeling languages for modeling software systems and model transformations for synthesizing them. Model transformations are in many ways similar to traditional artifacts, i.e., they have to be used by multiple developers, have to be changed according to changing requirements and should preferably be reused. Therefore, it is necessary to define and assess their quality. Quality attributes such as modifiability, understandability and reusability need to be understood and defined in the context of MDE, in particular for model transformations. For most other types of software artifacts, e.g. source code and models, there already exist approaches for measuring their quality. The goal of our research is to make the quality of model transformations measurable. In this paper, we focus on model transformations created using the ASF+SDF [2] term

* This work has been carried out as part of the FALCON project under the responsibility of the Embedded Systems Institute with Vanderlande Industries as the industrial partner. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Embedded Systems Institute (BSIK03021) program.

rewriting system which is actively applied in MDE projects [3,4,5]. However, we expect that our approach can be generalized and applied to other model transformations formalisms such as ATL [6], QVT [7] and openArchitectureWare [8].

We propose the following six quality attributes for measuring the quality of model transformations: *understandability*, *modifiability*, *reusability*, *modularity*, *completeness*, and *consistency*. Most of these quality attributes have already been defined earlier for software artifacts in general [9]. In [10], we explain why they are relevant for model transformations in particular. We also define 27 metrics for ASF+SDF transformations as predictors for these quality attributes. Some of these metrics are specific for ASF+SDF only, but for most metrics a conceptually equivalent metric can be defined in other model transformation formalisms as well. To assess whether the metrics are valid predictors for the quality attributes, an empirical analysis has been conducted. Metrics have been collected from a total of six transformations by a tool we created. The same cases have also been manually assessed by ASF+SDF experts. We correlate the metrics data with the expert data to explore the relations between the metrics and the quality attributes. In this way we can assess whether the automatically collected metrics are appropriate predictors for the quality attributes.

The remainder of this paper is structured as follows. In Section 2, the metrics we define to predict the quality attributes are described. Section 3 describes the results of our empirical study. Section 4 describes related work. Conclusions and directions for further research are given in Section 5. Note that we will not describe ASF+SDF here. Instead, the reader is referred to [2]. For an extended version of this work the reader is referred to [10].

2 Metrics

This section describes the metrics we defined and measure with a tool we implemented. The metrics described here are specific for ASF+SDF. However, for most of them a conceptually equivalent metric can be defined for other model transformation formalisms as well. All metrics are listed in Table 2.

2.1 Transformation Function Metrics

A measure for the size of a model transformation is the number of transformation functions it encompasses. A transformation function in ASF+SDF consists of one or more signatures and one or more equations. The *number of transformation functions* is therefore defined as the number of signatures that are implemented by at least one equation. The size of individual transformation functions can be measured by the metrics *number of signatures per function* and *number of equations per function*. These metrics measure the number of variants of a transformation function. Equations may have conditions. We measure the size of an equation as the *number of conditions* it has. Conditions can also be included when measuring the size of a transformation function. This leads to the metric *number of equations and conditions per function*. In this case, the number of variants of a transformation function is measured along with their sizes.

Measurements for the complexity of a transformation function are the number of arguments it takes and the number of values it returns. In ASF+SDF, a transformation function can be overloaded by defining multiple signatures and equations for it. These signatures may have different arguments. We measure the average number of arguments of a transformation function. This metric is called *val-in*. In ASF+SDF, a transformation function can return only one value. Therefore it does not make sense to measure the number of return values of a transformation function, i.e., val-out. However, different signatures of an overloaded transformation function may return values of different types. Therefore, we measure the *number of distinct return types per function*.

Transformation functions generally depend on other transformation functions. To measure this dependency, we measure *fan-in* and *fan-out* of transformation functions. Fan-in of a transformation function f is the number of times f is invoked by another transformation function f'. Fan-out of a transformation function f is the number of times f invokes another transformation function f'.

In ASF+SDF, there are a few mechanisms to influence the flow of control of the transformation engine. These are, amongst others, conditions, default equations and traversal functions. Two types of conditions can be distinguished, viz. matching conditions and (in)equality conditions. We measure how often the different condition types are used, by measuring the *number of matching conditions per equation* and the *number of (in)equality conditions per equation*. The number of matching conditions is of particular interest. It is possible to write equations that express the same in different ways. One can either write relatively small equations with a relatively large number of matching conditions, or relatively large equations with relatively few matching conditions. Upon evaluation, default equations are always evaluated last. Transformation functions without a default equation may be incomplete and hence may not rewrite properly. Therefore, we measure the *number of default equations per function*. Traversal functions can also be used to change the way evaluation of a transformation function is performed. A traversal function visits every node of a tree once, whereas a standard transformation function is applied to one node only. In this way, traversal functions allow a collapse of the number of transformation functions corresponding to a syntax directed translation scheme. Therefore we distinguish *traversal functions* when measuring the number of transformation functions. In other transformation formalisms, different mechanisms are used to influence the transformation engine. For example, in ATL it is possible to use lazy matched rules. A standard matched rule is applied only once, whereas a lazy matched rule is applied as often as it is referred to [6].

2.2 Module Metrics

Most model transformation formalisms enable a modular definition of model transformations. This is also the case for ASF+SDF. The *number of modules* is a measure for the size of a model transformation. The size of an individual module can be measured in different ways. We introduce three metrics to measure the size of a module, viz. the *number of transformation functions per module*, the *number of signatures per module* and the *number of equations per module.* These metrics can be compared with the average values over all modules to assess the balance of a module with respect to the rest of the model transformation.

Dependencies between modules can be measured on a module level. A module m depends on another module m' if module m imports module m'. To measure this type of dependency between modules, we measure the *number of import declarations per module* and the *number of times a module is imported* by other modules. Dependencies between modules can also be measured on the level of transformation functions. Transformation functions may invoke transformation functions defined in other modules. To measure this type of dependency between modules, we measure *fan-in* and *fan-out* for modules. Fan-in of a module m is the number of times a transformation function defined in module m is invoked by a transformation function defined in another module m'. Fan-out of a module m is the number of times a transformation function defined in module m invokes a transformation function defined in another module m'.

2.3 Consistency Metrics

A transformation function in ASF+SDF consists of signatures and equations. Each signature is related to one or more equations. A signature may have no related equations. This can for instance occur when a transformation is still under development. To detect this inconsistency, we measure the *number of signatures without equations.* An equation that is not related to a signature will be detected by ASF+SDF itself. Therefore we do not measure this.

Variables are usually defined in a `hiddens` section. This means that they can only be used in the module they are defined in. Therefore, a variable needs to be redefined if it is to be used in other modules. This may lead to inconsistencies in variable naming, i.e., a variable name in one module can be related to a different type in another module, or vice versa. It may also cause (re)definition of variables that are not used in a module. To detect these inconsistencies, we measure the *number of (different) variable names per type*, the *number of (different) types per variable name* and the *number of unused variables.* A variable is unused in a module if there are no instances of it used in the module it is defined in.

3 Empirical Exploration of the Metrics

The quality attributes relevant for the evaluation of model transformations in practice are not directly measurable. Therefore, we are interested in the relation between metrics and quality attributes. The purpose of the case study described in this section is to explore this relation. In the case study, we used six model transformations specified in ASF+SDF. For each of these transformations we collected metrics data. To evaluate the quality attributes for each of the transformations directly, we used a questionnaire that was completed by four experts in ASF+SDF. In this section we describe the design of the case study and the statistical analysis and interpretation of the collected data.

3.1 Objects, Subjects, Task and Instrumentation

The experimental objects are six model transformations specified in ASF+SDF. These transformations are real-world transformations created by different developers in research projects. The transformations differ in size, style, structure and functionality. Table 1 summarizes the characteristics of the transformations.

Table 1. Characteristics of the analyzed model transformations

Transformation	LOC	# Functions	Purpose	Reference
ACP2UML	5694	173	Transform process algebra models into UML	[3]
SL2XMI	1851	70	Transform surface language into activities	[4]
SLCheck	1430	58	Surface language wellformedness checker	[4]
ASF2C	7096	396	Generate C code from ASF specifications	[11]
UML2DOT	1553	28	Transform UML activities into the DOT language	–
REPLEO	4058	47	Syntax-safe template engine	[5]

The subjects in the study were four experienced users of ASF+SDF. All subjects are researchers who have developed several ASF+SDF transformations. None of the authors participated as subject in this study. Prior to their task, the subjects were not informed about the particular purpose of the study. Their task was to answer a questionnaire consisting of 23 questions. The questionnaire contained at least three similar, but different questions for each of the quality attributes. In each question, the subjects had to indicate their evaluation of one of the quality attributes on a five-point Likert scale (1 indicating a very low value and 5 indicating a very high value). The questionnaire can be found in [10]. For each of the six transformations, the subjects used the same questionnaire. Five transformations were evaluated by three subjects, the transformation "ASF2C" was evaluated by all four subjects. During the evaluation, the subjects had the transformation opened in the ASF+SDF Meta-Environment [12] on their own computer. There was no time-bound for the evaluation task. In addition to the quantitative evaluation of each of the transformations, a semi-structured interview was conducted after the questionnaire task to obtain qualitative statements.

The metrics were collected using the metrics collection tool we implemented. We collected the data without taking library modules into account since library modules can severely affect the analysis results.

3.2 Relating Metrics to Quality Attributes

To establish the relation between metrics and quality attributes we analyze the correlation between them. The data acquired from the questionnaire is ordinal. Therefore, we use a non-parametric rank correlation test [13]. Since the data set is small and we expect a number of tied ranks, we use Kendall's τ_b rank correlation test [14]. This test returns two values, viz. significance and correlation coefficient. The significance indicates the probability that there is no correlation between metric and quality attribute even though one is reported, i.e., the probability for a coincidence. Since we are performing an exploratory study and not an in-depth study, we accept a significance level of 0,10. The correlation coefficient indicates

Table 2. Kendall's τ_b correlations

		Understandability		Modularity		Modifiability		Reusability		Completeness		Consistency	
#	Metric	CC	Sig.	CC	Sig.	CC	Sig.	CC	Sig.	CC	Sig.	CC	Sig.
1	# Transformation functions	-,550	,002	,439	,017	-,303	,092	-,242	,183	,053	,772	-,190	,307
2	# Traversal Functions	-,356	,048	,159	,385	-,432	,016	-,216	,235	,053	,772	-,136	,465
3	# Modules	-,801	,000	,713	,000	-,553	,003	-,256	,166	-,164	,379	-,483	,011
4	# Signatures without equations	-,481	,009	,631	,001	-,146	,424	-,135	,466	-,150	,420	-,476	,012
5	# Functions per module	-,278	,123	,279	,128	-,032	,858	-,111	,541	,106	,563	-,054	,770
6	# Traversal functions per module	-,006	,971	-,120	,515	-,058	,747	-,163	,369	,093	,613	,149	,422
7	# Signatures per module	-,123	,495	,120	,515	,123	,496	-,059	,746	,106	,563	,027	,884
8	# Equations per module	,667	,000	-,638	,001	,381	,035	,177	,331	,212	,247	,570	,002
9	# Signatures per function	,032	,858	,040	,828	,019	,914	-,033	,857	-,172	,347	-,203	,274
10	# Equations per function	-,265	,141	,279	,128	-,213	,237	,033	,857	-,146	,426	-,285	,125
11	# Default equations per function	,084	,641	-,080	,664	-,161	,370	,098	,590	-,053	,772	-,041	,827
12	# Eqs. and conditions per function	,175	,332	-,093	,612	,252	,162	,229	,208	-,040	,828	-,068	,715
13	Function fan-in	-,136	,451	,106	,562	-,058	,747	,111	,541	-,040	,828	-,176	,343
14	Function fan-out	,019	,914	,066	,717	,097	,591	,177	,331	-,040	,828	-,149	,422
15	Module fan-in	-,601	,001	,690	,000	-,307	,104	-,195	,306	-,073	,703	-,374	,054
16	Module fan-out	,188	,298	-,199	,277	,432	,016	,059	,746	,106	,563	,271	,144
17	# Conditions per equation	,550	,002	-,518	,005	,432	,016	,190	,297	,238	,193	,488	,009
18	# Matching conditions per equation	,693	,000	-,571	,002	,587	,001	,268	,140	,172	,347	,407	,029
19	# Equality conditions per equation	-,278	,123	,372	,043	-,097	,591	-,072	,692	,026	,885	-,068	,715
20	# Distinct return types per function	-,537	,003	,465	,011	-,355	,049	-,150	,408	-,093	,613	-,515	,006
21	# Import declarations per module	-,162	,370	,319	,082	,148	,410	-,059	,746	-,172	,347	-,258	,165
22	# Times a module is imported	-,655	,000	,658	,000	-,393	,032	-,175	,343	-,232	,213	-,566	,003
23	# Variables per type	-,758	,000	,678	,000	-,432	,016	-,268	,140	-,053	,772	-,407	,029
24	# Distinct variables per type	-,758	,000	,678	,000	-,432	,016	-,268	,140	-,053	,772	-,407	,029
25	# Types per variable	-,658	,000	,631	,001	-,378	,045	-,152	,426	-,190	,321	-,584	,003
26	# Unused variables per module	-,291	,106	,332	,070	-,071	,694	-,124	,494	,053	,772	-,054	,770
27	Average val-in	-,123	,495	,080	,664	-,136	,452	-,098	,590	-,172	,347	-,231	,215

CC: Correlation coefficient
Sig.: Two-tailed significance

the strength and direction of the correlation. A positive correlation coefficient means that there is a positive relation between metric and quality attribute and a negative correlation coefficient implies a negative relation. Note that correlation does not indicate a causal relation between metric and quality attribute. Table 2 contains the correlations we acquired. The significant correlations are marked.

No metric correlates significantly with reusability. The reason for this is that the experts cannot evaluate reusability properly because they do not see what they can reuse parts of the transformations for. Also, no metric correlates significantly with completeness. The reason for this is that the experts could not evaluate completeness properly because they did not have the specification of the analyzed transformations. Moreover, the time needed to get acquainted with the source and target language of the transformations is large.

The metrics that indicate the size of a transformation, i.e., *number of (traversal) functions* and *number of modules* correlate negatively with both understandability and modifiability. This indicates that larger model transformations are harder to understand and to modify. The same size metrics correlate positively with modularity. In larger transformations the need for splitting functionality over modules becomes higher. Therefore, a larger transformation often implies more modules, and therefore a more modular transformation. However, a high number of modules alone is not enough for a model transformation to be modular. Functionality should be well-spread over these modules.

The metric *number of modules* correlates negatively with consistency. More modules often implies a more complex transformation and more interfaces between modules. This may lead to inconsistencies. Also, when multiple developers work on a transformation it is likely that they work on separate modules. Since every developer has his own style, this may lead to inconsistencies.

The *number of (matching) conditions per equation* is positively correlated with understandability and modifiability. When writing equations, a tradeoff has to be made between writing a complex equation with little matching conditions or writing a simple equation with more matching conditions. The correlation indicates that simple equations with more matching conditions are preferred.

The *number of equations per module* correlates negatively with modularity. Modularity means that functionality should be spread over modules. This usually leads to smaller modules, i.e., modules with fewer equations.

In transformations consisting of multiple modules, modules depend on each other. This is expressed by module fan-in, module fan-out, the number of times a module is imported, and the number of import declarations. Therefore, *module fan-in* and *number of times a module is imported* correlate significantly in a positive way with modularity. These two metrics correlate negatively with modifiability. When a module on which other modules depend needs modifications, attention should be paid that these dependencies remain correct.

The *number of distinct return types per function* correlates negatively with modifiability and consistency. A function with multiple return types has multiple equations. This has two disadvantages with respect to modifying a transformation. First, if only one, or a few equations need modifications, attention should be paid that the correct equation is modified. Second, more equations imply more modifications. The correlation with consistency is to be expected because the return types are not consistent with each other.

The *number of types per variable* also correlates with consistency negatively. This is to be expected, because a variable that is of a different type in different modules is inconsistently defined. Related to this is the negative correlation between the number of (distinct) variables per type on consistency. Redefinition of variables may lead to inconsistent naming. In fact, the metric *number of (distinct) variables per type* measures this directly.

3.3 Threats to Validity

Conducting empirical studies involve threats to validity. Here we discuss how we addressed potential threats to validity in the presented study.

An important issue that must be taken into account for empirical studies is the representativeness of the experimental design with respect to practice.

We selected experienced ASF+SDF experts as subjects in our study. Since our experience shows that model transformations are developed and maintained by experts in practice, we exclude the subject experience as a threat to the validity in our study. The transformations used as objects in our study are designed in and applied for practical purposes. Additionally, our sample of transformations is heterogeneous with respect to several characteristics. Hence, we do not consider the object selection as a threat to the validity of this study.

Our choice for the transformation formalism ASF+SDF could be discussed. Future replications of our study must prove whether the findings for ASF+SDF presented in this study will also hold for conceptually similar metrics for other transformation formalisms. In our study the objects conducted an evaluation task that is not representative for practical model engineering tasks, and therefore this is a potential threat to the validity. We addressed this threat in the design of our study by using at least three self-controlled questions for each quality attribute and we used the evaluations of four experts. The results were relatively consistent between the experts and between the self-controlled questions, respectively. Therefore we minimized this threat to validity.

The number of observations in this study is rather small. This is a potential threat to the validity. It is difficult to find a larger number of experts in ASF+SDF for participation in such a study. We accepted this threat to the validity, because the study is only a first exploration of transformation quality metrics.

4 Related Work

The authors of [15] discuss characteristics of MDE that should be taken into account when developing a quality framework for it. They also define a quality framework for MDE themselves. Like us, the authors recognize the need for evaluating the quality of model transformations. Therefore, they apply their framework to this matter. This results in a set of quality attributes and a suggested method for assessing them. Our approach complements theirs. The quality attributes we propose include the ones they propose for model transformations. In addition, we present metrics for assessing the quality attributes we defined.

In [16], a set of metrics is proposed to monitor iterative grammar development in SDF. The authors took metrics developed by others that are applicable to measure (E)BNF grammars and adapted them such that they can be used to measure SDF grammars. The difference with our work is that they focus on grammar development using SDF, whereas we focus on transformation development using both ASF and SDF.

In [17], metrics are defined for functional programming languages. Since ASF is a functional language, we were able to adapt some of the metrics they defined such that they can be used to measure the quality of model transformations.

We assess the relation between metrics and quality attributes empirically. Multiple experiments that use a similar approach are described in [18].

5 Conclusions and Future Work

5.1 Conclusions

We have addressed the necessity for a methodology to analyze the quality of model transformations. In this paper, we proposed six quality attributes to evaluate the quality of model transformations. We also defined a set of 27 metrics for predicting these quality attributes for model transformations created using ASF+SDF. These metrics can automatically be collected from ASF+SDF model transformation specifications by a tool we created.

For the evaluation of quality attributes of model transformations, it is necessary to be able to select appropriate metrics as indicators for the quality attributes. Our study is a first step into this direction and provides data that supports the selection of metrics for particular quality attributes. For most of the proposed quality attributes we found metrics that correlate with them. This can, amongst others, be used to indicate possible points for improvements in model transformations.

5.2 Future Work

The manual assessment of the quality of the model transformations was carried out by four ASF+SDF experts. In the future, we would like to have feedback from more experts and on more cases. In that way, the results will be more significant. Also, we would like to perform a more in-depth statistical analysis.

In this paper, we focused on model transformations created using ASF+SDF. We expect that our techniques can be generalized and applied to different model transformation formalisms as well. Our focus will be on ATL [6]. The quality attributes will be the same, but the metrics to predict the quality attributes will differ. However, we expect that most metrics will be conceptually similar.

Once we have identified quality problems in model transformations, we can propose a methodology for improving their quality. This methodology will probably consist of a set of guidelines which, if adhered to, lead to high-quality model transformations.

Acknowledgements. We want to thank Mark Ligtvoet for his help with implementing the tool and Jan Stoop for his help with the statistical analysis. We also want to thank the ASF+SDF experts for participating in this research.

References

1. Schmidt, D.C.: Model-driven engineering. Computer 39(2), 25–31 (2006)
2. van Deursen, A.: An overview of ASF+SDF. In: Language Prototyping: An Algebraic Specification Approach. AMAST Series in Computing, vol. 5, pp. 1–29. World Scientific Publishing, Singapore (1996)
3. van Amstel, M.F., van den Brand, M.G.J., Protić, Z., Verhoeff, T.: Transforming process algebra models into UML state machines: Bridging a semantic gap? In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 61–75. Springer, Heidelberg (2008)

4. Engelen, L.J.P., van den Brand, M.G.J.: Integrating textual and graphical modelling languages. In: Proceedings of the 9th Workshop on Language Descriptions, Tools and Applications (2009)
5. Arnoldus, B.J., Bijpost, J., van den Brand, M.G.J.: REPLEO: a syntax-safe template engine. In: Proceedings of the 6th international conference on Generative programming and component engineering, pp. 25–32. ACM, New York (2007)
6. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
7. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation specification. OMG Document formal/2008-04-03, OMG (2008)
8. Völter, M.: OpenArchitectureWare: a flexible open source platform for model-driven software development. In: Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference (2006)
9. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., Macleod, G.J., Merrit, M.J.: Characteristics of Software Quality. North-Holland, Amsterdam (1978)
10. van Amstel, M.F., Lange, C.F.J., van den Brand, M.G.J.: Evaluating the quality of ASF+SDF model transformations. CS-report, Eindhoven University of Technology, Eindhoven, The Netherlands (2009)
11. van den Brand, M.G.J., Heering, J., Klint, P., Olivier, P.A.: Compiling rewrite systems: The ASF+SDF compiler. ACM Transactions on Programming Languages and Systems 24(4), 334–368 (2002)
12. van den Brand, M.G.J., van Deursen, A., Heering, J., de Jong, H.A., de Jonge, M., Kuipers, T., Klint, P., Moonen, L., Olivier, P.A., Scheerder, J., Vinju, J.J., Visser, E., Visser, J.: The ASF+SDF meta-environment: A component-based language development environment. In: Wilhelm, R. (ed.) CC 2001. LNCS, vol. 2027, pp. 365–370. Springer, Heidelberg (2001)
13. Fenton, N.E., Pfleeger, S.L.: Software Metrics: A Rigorous & Practical Approach, 2nd edn. PWS Publishing Co. (1996)
14. Field, A.: Discovering Statistics using SPSS, 2nd edn. Sage, Thousand Oaks (2005)
15. Mohagheghi, P., Dehlen, V.: Developing a quality framework for model-driven engineering. In: Giese, H. (ed.) MODELS 2008. LNCS, vol. 5002, pp. 275–286. Springer, Heidelberg (2008)
16. Alves, T., Visser, J.: Metrication of SDF grammars. Technical report, Departamento de Informática da Universidade do Minho, Braga, Portugal (2005)
17. Harrison, R.: Quantifying internal attributes of functional programs. Information and Software Technology 35(10), 554–560 (1993)
18. Lange, C.F.J.: Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML. Ph.D thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (2007)