Technical debt in Model Transformation specifications

K. Lano, S. Kolahdouz-Rahimi, M. Sharbaf

Dept. of Informatics, King's College London
Email: kevin.lano@kcl.ac.uk
Dept. of Software Engineering
University of Isfahan, Iran
Email: { sh.rahimi, m.sharbaf }@eng.ui.ac.ir

Abstract. Model transformations, as with any other software artifact, may contain quality flaws. Even if a transformation is functionally correct, such flaws will impair maintenance activities such as enhancement and porting. The concept of *technical debt* (TD) models the impact of such flaws as a burden carried by the software which must either be settled in a 'lump sum' to eradicate the flaw, or paid in the ongoing additional costs of maintaining the software with the flaw. In this paper we investigate the characteristics of technical debt in model transformations (MT), analysing a range of MT cases in different MT languages, and using appropriate measures of quality flaws or 'bad smells' for MT, adapted from code measures.
We identify significant differences in the level and kinds of technical debt in different MT languages, and we propose ways in which the TD level can be reduced.

1 Introduction

This paper will investigate the issue of *technical debt* (TD) [24] in MT. Technical debt refers to the short and long-term impact of software quality flaws such as duplicated code. The *principal* cost of TD is incurred when refactoring or other redesign is used to remove the TD from the software, whilst the *interest* is paid in the additional cost due to the TD each time the software is maintained.

The concept of TD was initially applied to code artifacts, but can also be extended to analysis and design models [3].

In the MDE context, model transformations are a key software resource, which enable MDE processes such as the production of software and documentation from models, the synchronisation of models, and model comparison. Thus the quality and maintainability of MT are likely to be important factors in the successful use of MDE. Our own experience with large code generators [20] confirms this, and the issue of MT maintenance is also evident in some of the industrial MT cases of [27].

The high-level goal of our research is to *quantify, and characterise the nature of, technical debt in model transformations development.* We will adopt the goal-question-metric (GQM) approach of [4] to decompose this goal into specific questions and metrics. The goal leads to the following research questions:

RQ1: What is the extent of TD in MT cases?
RQ2: What are the most frequent forms of quality flaw in MT cases?
RQ3: Does the level and character of TD vary between MT languages and between MT categories?
RQ4: Is there a difference between TD in MT languages and TD in traditional programming languages?

The questions imply that a significant sample of transformations must be surveyed, for a range of transformation languages and categories. We will use published and machine-readable transformation cases, and public repositories of transformations. Only cases where the complete code of the transformations is available will be considered. We survey the ATL and QVT-R transformation languages because these are the most widely-used MT languages by practitioners [**?**]. We also consider ETL and UML-RSDS, which are MT languages with distinctive features whose impact on TD levels is of interest.

2 Metrics for technical debt

Following on from the research questions, we need to find concrete measures which quantify the aspects (TD and categories of TD) which the questions refer to. Measures of various 'bad smells' or quality flaws are typically used as metrics of TD in code. However, these need adaption when used for declarative or hybrid MT specification languages: MT specifications define their effect in a less procedural manner than code, they are usually more concise, and do not usually have a direct concept of class or association. Thus the threshold values for measures of size will usually be lower than corresponding code measures [8].

Therefore we define measures specific to MT specifications, adapting TD measures *Excessive Class Length*, *Excessive Method Length*, *Excessive Parameter Length*, *Duplicate Code*, *Cyclomatic Complexity*, *Coupling Between Objects*, *Too Many Methods* to the MT context.

Based on our experience of developing and maintaining MT specifications, we considered that the following were the most significant factors in impeding the understanding and maintenance of MT specifications: size; semantic complexity (of expressions, rules and operations); complexity of relationships and dependencies between rules/operations; redundancy. These impact the Analysability, Changeability and Testability quality characteristics of software as defined in the ISO/IEC 9126-1 quality model [9]. In practice they manifest as:

- Excessively large transformations, with many rules/operations and/or high total length (MT size factor)
- Unclear rule precedence or execution order (MT rule dependency factor)
- Excessively complex expressions (MT semantic complexity factor)
- Excessive rule or operation length (MT size factor)

– Excessive numbers of parameters/auxiliary variables for a rule, transformation or operation (MT semantic complexity factor)
– Duplicated expressions or code (MT redundancy factor)
– Complex rule or code logic (MT semantic complexity factor)
– Complex calling relations between rules, especially cyclic relations (self or mutual recursion). Inheritance of rules/operations is also counted as a dependency of the generalised rule/operation upon the specialised rules/operations. (MT rule dependency factor)
– Excessive numbers of rules/operations called from one rule or operation (MT rule dependency factor).

The size of software artifacts is often measured in terms of lines of code (LOC). However, LOC in different software languages are not comparable, because of the differing syntax and density of functionality in different languages. Thus, typically, a line of a MT language such as ATL will contain more functionality than a line of Java. We will investigate how LOC correlates with other measures of size. In particular, a measure $c(\tau)$ of the semantic content of a model transformation specification τ will be used, based on the complexity of expressions/activities in the transformation: the total number of operators and identifiers in OCL constraints or in activities/statements. Each of ATL, ETL, QVT-R and UML-RSDS have similar expression languages based on OCL, and ATL, ETL and UML-RSDS have similar activity languages. Table 1 summarises the semantic size measure $c(e)$ for OCL expressions e. $c(e)$ can be considered a count of the number of basic semantic elements in a specification (identifiers plus operators). We also include a token count measure $t(e)$, which is used for clone detection.

Expression e	*Complexity* $c(e)$	*Token count* $t(e)$
Numeric, boolean or String value	0	1
Identifier *iden* Basic expression $obj.f$	1 $c(obj) + c(f) + 1$	1 $t(obj) + t(f) + 1$
Operation call $e(p1, ..., pn)$	$c(e) + 1 + \Sigma_i c(pi)$	$t(e) + n + 1 + \Sigma_i t(pi)$
Unary expression $op\ e$ $e \rightarrow op()$	$1 + c(e)$	$1 + t(e)$ $4 + t(e)$
Binary expression $e1\ op\ e2$ $e1 \rightarrow op(e2)$	$c(e1) + c(e2) + 1$	$t(e1) + t(e2) + 1$ $t(e1) + t(e2) + 4$
Ternary expression $op(e1, e2, e3)$ *if* $e1$ *then* $e2$ *else* $e3$ *endif*	$c(e1) + c(e2) + c(e3) + 1$	$t(e1) + t(e2) + t(e3) + 5$ $t(e1) + t(e2) + t(e3) + 4$
$Set\{e1, ..., en\}$ $Sequence\{e1, ..., en\}$	$1 + \Sigma_i c(ei)$	$2 + n + \Sigma_i t(ei)$

Table 1. OCL expression complexity measures

The $c(e)$ measure is related to the effort needed to comprehend an expression [5]. For example, to understand the expression $E \rightarrow exists(P)$, a MT developer needs to understand E, P and the effect/meaning of the quantifier operator. The measure of rule size adopted in [26] is also based on the expression complexity: they count nodes, edges and attributes in a graph pattern, corresponding to objects, association navigations and attribute evaluations in OCL.

A similar measure can be given to activities (Table 2 shows the values for UML-RSDS syntax, similar definitions can be given for the ATL and ETL statement syntax).

Activity s	*Complexity* $c(s)$	*Token count*
`return` e	$1 + c(e)$	$1 + t(e)$
$v := e$	$c(v) + c(e) + 1$	$t(v) + t(e) + 1$
$s1;\ s2$	$c(s1) + c(s2) + 1$	$t(s1) + t(s2) + 1$
Operation call $e(p1, ..., pn)$	$c(e) + 1 + \Sigma_i c(pi)$	$t(e) + n + 1 + \Sigma_i t(pi)$
`if` e `then` $s1$ `else` $s2$	$1 + c(e) + c(s1) + c(s2)$	$3 + t(e) + t(s1) + t(s2)$
`for` $v : e$ `do` s	$c(e) + c(s) + 1$	$3 + t(e) + t(v) + t(s)$
`while` e `do` s	$c(e) + c(s) + 1$	$t(e) + t(s) + 2$
`break`	1	1
`continue`	1	1

Table 2. Activity complexity measures

Using these measures, $c(r)$ for a transformation rule r is taken as the sum of the c measures of its parts (such as *from*, *to* and *do* clauses in ATL), likewise for operation definitions. The semantic complexity $c(\tau)$ of a transformation is taken as the sum of the complexities of its rules and operations. We also adopt the metric of *fan-out* from [10], this is the number of different rules or operations called from one rule or operation. This quantity has a direct impact on the understandability of the calling rule/operation.

We also consider LOC measures of size because of the prevalence of this measure in TD estimation. We will evaluate flaw density both wrt LOC and complexity. Based on [10], we adopt 50 LOC per rule/operation and 500 LOC per transformation as size thresholds, for size measured by LOC. These thresholds apply to ATL, ETL and QVT-R. For UML-RSDS we adopt limits based on expression complexity (100 and 1000 respectively) since UML-RSDS is based on graphical use cases and class diagrams, and does not have a standard text representation. These limits are based on our experience with maintenance of UML-RSDS transformations.

Technical debt in MT developments will therefore be measured by identifying the frequency of occurrence of the following specific 'bad smells' in MT specifications:

ETS: Excessive transformation size ($c(\tau) > 1000$, or length ¿ 500 LOC)
ENR: Excessive number of rules ($nrules > 10$)

ENO: Excessive number of helpers/operations ($nops > 10$)
UEX: Excessive use of undefined execution orders/priorities between rules ($>$ 10 undefined orderings)
ERS: Excessive rule size ($c(r) > 100$ or length greater than 50 LOC)
EHS: Excessive helper size ($c(h) > 100$ or length $>$ 50 LOC)
EPL: Excessive parameter list (for transformation, rules, and helpers): $>$ 10 parameters including auxiliary rule/operation variables
DC: Duplicate expressions/code (duplicate expressions or statements x with token count $t(x) > 10$)
CC: Cyclomatic complexity (of rule logic or of procedural code) ($>$ 10)
CBR: Coupling between rules (number of rule/operation explicit or implicit calling relations $> nrules + nops$, or any cyclic dependencies exist in the rule/operation call graph).
EFO: Excessive fan-out of a rule/operation ($>$ 5 different rules/operations called from one rule/operation).

Number of tokens is used for detecting clones, because in this case value expressions should be counted as contributing to the clone. The lower limit for clone size is set to avoid trivial clones. It could be reduced, at the cost of increased processing time. In [26] clones of any size are considered. In [8], a lower bound of 50 tokens is used for code clone detection. We experimented with using 25 tokens as the threshold, but this led to many significant clones being ignored, and we adopted 10 tokens for our analysis.

Note that if a transformation has completely undefined rule orders, then it cannot contain more than 5 rules without breaking the limit for undefined rule orders ($UEX = \frac{n*(n-1)}{2}$ in this case, where n is the number of rules).

At present, we limit our scope to considering individual transformations, rather than transformations in a system of inter-operating transformations. The coupling between different transformations in such a system could also be evaluated, together with the number of transformations and their calling/dependency relations. We also do not consider problematic issues in the use of OCL [6] – OCL 'smells' such as the use of chained *implies*, 'magic literals', chained *forAll* quantifiers, long chained navigations in expressions, and other constructions which impair the comprehensibility of the specification. To the issues of [6] we would add problems such as the use of general *iterate* expressions, or explicit use of the *invalid* value. OCL flaws are a semantic complexity factor for which specific metrics could be devised.

3 Results

The measures of TD are computed on the abstract syntax representations of ATL, ETL, QVT-R and UML-RSDS specifications, according to the respective metamodels of these languages. The languages have many similarities at this level (eg., top-level rules in QVT-R correspond to non-lazy rules in ETL and to use case constraints in UML-RSDS). Hence the same general specification of measures can be applied to each language, with some differences to account for the different language styles and semantics.

3.1 ATL

For ATL we consider the cases of Table 3 from the ATL transformations Zoo, which is widely used in surveys of model transformations. The cases are chosen as being typical of medium to large sized ATL transformations. Size measures are based on LOC, with 50 LOC being considered the threshold for operation and rule size, and 500 lines for transformations. We show separately the LOC measures *rs* of the transformation rules and *os* of the helper operations, after their total. *ENR* is the number of rules in the case, *ENO* is the number of operations. *ERS* is the number of rules with length over the threshold (50 LOC), likewise *EHS* for operations. *CC* is the number of rules/operations over the *CC* threshold (10). For *CBR*, N(M) is the number N of rule/operation dependencies and the number M of rules/operations which occur in cycles of calling dependencies. *DC* is the number of distinct cloned expressions (e with $t(e) > 10$) in the case. *UEX* is $n * (n-1)/2$ where n is the number of concrete non-lazy, non-called rules. Underlined measures identify where flaws occur.

Table 3 gives the measures for the ATL Zoo cases. For all of the examples *EPL* and *EFO* are 0, so are omitted. Where a transformation consists of several subtransformations, we list these as (i), (ii) etc below the main transformation entry.

Table 4 gives a summary of the technical debt of these cases. To compute the number of flaws in a transformation, we count 1 for each of *ETS*, *ENR*, *ENO*, *UEX*, *CBR* over the thresholds, plus *ERS* + *EHS* + *CC* + *EPL* + *EFO* + *DC*, plus the number of self-dependent rules/operations. For a transformation system, we sum the number of flaws in each of its subtransformations. It is noticeable that the number of flaws per LOC is quite similar across all of the cases, and the s.d. is 0.0023. In addition, although ATL is well-known for emphasising helper functions to perform transformations, in all of these cases the majority of code is in rules.

The correlation between LOC and number of flaws is 0.915, which is significant at the 5% level [29], indicating a strong linear correlation between LOC and number of flaws. It can be noted that the ratio of complexity to LOC is 1.71, reflecting the relatively low semantic density of typical ATL specifications. The flaw rate per semantic element is 0.00931 (number of flaws divided by complexity).

3.2 ETL

ETL has a similar rule and transformation structure to ATL, but with a more general processing model and more complex semantics. For ETL we define UEX as $\frac{n*(n-1)}{2}$ where n is the number of concrete non-lazy rules. We identified ETL cases to analyse from the Eclipse ETL repository (git.eclipse.org), and from other published cases (github.com/epsilonlabs).

ETL has implicit invocation of rules by rules or operations, where the text of the transformation does not contain an explicit reference to rules that may be invoked due to *equivalent*/*equivalents* expressions. In calculating the call graph

Transformation	*ETS* (rs, os)	*ENR*	*ENO*	*ERS*	*EHS*	*CC*	*CBR*	*DC*	*UEX*
MOF to UML	$\underline{935}$ (746, 189)	$\underline{11}$	$\underline{11}$	$\underline{5}$	0	0	$\underline{27}$(0)	$\underline{7}$	$\underline{55}$
KM3 to DOT	451 (251,200)	7	$\underline{18}$	$\underline{1}$	0	0	$\underline{33}$(0)	$\underline{4}$	$\underline{21}$
MySQL to KM3	$\underline{995}$ (571, 424)	$\underline{20}$	$\underline{28}$	$\underline{1}$	0	$\underline{1}$	$\underline{62}$($\underline{4}$)	$\underline{7}$	$\underline{71}$
(i) *XML2XML*	101 (87, 14)	4	1	0	0	0	2(0)	$\underline{2}$	6
(ii) *XML2MySQL*	281 (137,144)	5	10	0	0	0	$\underline{22}$($\underline{2}$)	$\underline{2}$	10
(iii) *MySQL2KM3*	$\underline{613}$ (347,266)	$\underline{11}$	$\underline{17}$	$\underline{1}$	0	$\underline{1}$	$\underline{38}$($\underline{2}$)	$\underline{3}$	$\underline{55}$
Excel Injector	395 (231,164)	$\underline{11}$	10	0	0	0	$\underline{38}$(0)	$\underline{3}$	$\underline{55}$
Excel Extractor	311 (251,60)	$\underline{13}$	5	0	0	0	6($\underline{1}$)	$\underline{2}$	$\underline{66}$
(i) *SpreadsheetML Simplified2XML*	263 (246,17)	$\underline{12}$	1	0	0	0	1(0)	$\underline{2}$	$\underline{66}$
(ii) *XML2ExcelText*	48 (5,43)	1	4	0	0	0	5($\underline{1}$)	0	0
PetriNet to/from PathExpression	$\underline{1267}$ (799,468)	$\underline{23}$	$\underline{32}$	$\underline{2}$	$\underline{1}$	0	$\underline{88}$($\underline{2}$)	$\underline{8}$	$\underline{47}$
(i) *PetriNet2PathExp*	70 (70,0)	3	0	0	0	0	0(0)	$\underline{1}$	3
(ii) *XML2PetriNet*	228 (136,92)	5	8	0	0	0	$\underline{22}$(0)	$\underline{2}$	10
(iii) *PetriNet2XML*	222 (189,33)	5	3	$\underline{1}$	0	0	$\underline{12}$(0)	$\underline{4}$	10
(iv) *PathExp2PetriNet*	104 (87,17)	3	1	0	0	0	$\underline{5}$(0)	0	3
(v) *TextualPathExp2PathExp*	$\underline{643}$ (317,326)	7	$\underline{20}$	$\underline{1}$	$\underline{1}$	0	$\underline{49}$($\underline{2}$)	$\underline{1}$	$\underline{21}$
Make to Ant	368 (242,126)	$\underline{13}$	$\underline{11}$	0	0	0	13($\underline{2}$)	$\underline{2}$	$\underline{31}$
(i) *XML2Make*	147 (73,74)	5	7	0	0	0	7($\underline{1}$)	0	10
(ii) *Ant2XML*	177 (164,13)	7	1	0	0	0	2(0)	$\underline{2}$	$\underline{21}$
(iii) *XML2Text*	44 (5,39)	1	3	0	0	0	4($\underline{1}$)	0	0
Maven to Ant	$\underline{1307}$ (1139,168)	$\underline{90}$	$\underline{18}$	0	0	0	80(0)	$\underline{7}$	$\underline{1326}$
(i) *XML2Maven*	$\underline{575}$ (472,103)	$\underline{36}$	$\underline{13}$	0	0	0	$\underline{74}$(0)	$\underline{3}$	$\underline{630}$
(ii) *Maven2Ant*	360 (308,52)	$\underline{30}$	4	0	0	0	4(0)	$\underline{1}$	$\underline{420}$
(iii) *Ant2XML*	372 (359,13)	$\underline{24}$	1	0	0	0	2(0)	$\underline{3}$	$\underline{276}$

Table 3. Technical debt measures for ATL

Transformation	*Category*	*LOC*	$c(\tau)$	*% in rules*	*# flaws*	*flaws/LOC*
MOF to UML	Migration	935	1002	79.7%	17	0.018
KM3 to DOT	Refinement	451	926	55.6%	8	0.017
MySQL to KM3	Abstraction	995	1726	57.3%	19	0.019
Excel Injector	Migration	395	601	58.5%	6	0.015
Excel Extractor	Migration	311	528	81%	5	0.016
Petri Net from/to	Semantic map	1267	1645	63%	20	0.016
Make to Ant	Migration	368	808	65.7%	5	0.013
Maven to Ant	Migration	1307	3075	87%	16	0.012
Average		753.6	1288.9	70.2%	12	0.016

Table 4. Results summary for ATL

and *CBR* metric, such implicit calls must be taken into account. In ETL, an expression $e.equivalent()$ may implicitly invoke any concrete lazy or non-lazy rule which has an input variable $v : T$ with T containing the actual value of *e at runtime*. Thus the calling rule or operation implicitly depends upon all concrete rules in the transformation, potentially leading to large values for fan-out and call graph size. The abbreviated form $v ::= e$ of $v = e.equivalent()$ is considered in the same manner.

Table 5 gives the measures for the selected ETL cases. *CC* is omitted because it is 0 for all the cases. *EPL* is the number of rules/operations with more than 10 parameters, including local auxiliary variables. *EFO* is the number of rules/operations which depend on more than 5 rules/operations.

Transformation	*ETS* (rs, os)	*ENR*	*ENO*	*ERS*	*EHS*	*EPL*	*EFO*	*CBR*	*DC*	*UEX*
Flowchart2HTML	163 (163, 0)	19	0	0	0	0	0	10(2)	0	21
(i) *base*	24 (24,0)	4	0	0	0	0	0	0(0)	0	6
(ii) *equivalent*	21 (21,0)	2	0	0	0	0	0	1(0)	0	1
(iii) *greedy*	7 (7,0)	1	0	0	0	0	0	0(0)	0	0
(iv) *inheritance*	14 (14,0)	2	0	0	0	0	0	1(0)	0	1
(v) *lazy*	31 (31,0)	4	0	0	0	0	0	4(1)	0	6
(vi) *multipletargets*	32 (32,0)	2	0	0	0	0	0	0(0)	0	1
(vii) *primary*	34 (34,0)	4	0	0	0	0	0	4(1)	0	6
CopyFlowchart	57 (57,0)	5	0	0	0	0	0	25(5)	1	10
CopyOO	110 (110, 0)	10	0	0	0	0	9	90(9)	3	45
In2out	19 (19,0)	1	0	0	0	0	0	1(1)	0	0
OO2DB	142 (121,21)	6	3	0	0	0	1	20(4)	0	6
RSS2ATOM	88 (74,14)	9	2	0	0	0	2	20(2)	0	36
Tree2Graph	15 (15,0)	1	0	0	0	0	0	1(1)	0	0
uml2xsd	17 (17,0)	2	0	0	0	0	0	2(1)	0	1
MDDTIF	145 (139,6)	18	1	0	0	0	9	61(12)	1	39
(i) *Competition2TVPP*	30 (30,0)	3	0	0	0	0	0	3(1)	0	3
(ii) *CopyTVApp*	48 (48,0)	7	0	0	0	0	4	28(7)	1	21
(iii) *TVApp2Xml*	67 (61,6)	8	1	0	0	0	5	30(4)	0	15
Argouml2ecore	96 (76,20)	7	2	0	0	0	6	37(4)	1	21
StateElimination (TTC 2017)	313 (155,158)	8	2	1	2	2	0	5(2)	0	4
(i) *MainTask*	229 (126,103)	4	1	1	1	2	0	0(0)	0	1
(ii) *Extension1*	84 (29,55)	4	1	0	1	0	0	5(2)	0	3
TTC Live Case 2017	206 (163,43)	7	8	1	0	1	0	19(2)	0	2
(i) *Ecore2SimpleCodeDOM*	86 (50,36)	3	7	0	0	0	0	10(2)	0	0
(ii) *Ecore2SimpleCodeDOMA*	69 (69,0)	2	0	1	0	1	0	3(0)	0	1
(iii) *Ecore2SimpleCodeDOMB*	51 (44,7)	2	1	0	0	0	0	6(0)	0	1
uml2Simulink	148 (114,34)	7	6	0	0	0	5	39(4)	1	10

Table 5. Technical debt measures for ETL

Table 6 gives a summary of the technical debt of these cases. The same computation of number of flaws is used as for ATL. It is noticeable that the rate of flaws per LOC is higher than for ATL in general, and with a much wider range of rates than for ATL (the s.d. is 0.06). This may be due to the wide variety of styles supported by ETL, from the highly imperative transformations of *StateElimination*, to the very implicit and declarative *CopyOO*. In the most complex cases, such as *MDDTIF*, three forms of inter-rule/operation dependence are used simultaneously: inheritance, explicit calls and implicit calls, leading to high values for *CBR* and *EFO*.

Transformation	*Category*	*LOC*	$c(\tau)$	*% in rules*	*# flaws*	*flaws/LOC*
Flowchart2HTML	Code-generation	163	377	100%	2	0.012
CopyFlowchart	Migration	57	153	100%	7	0.122
CopyOO	Migration	110	438	100%	23	0.209
In2out	Migration	19	53	100%	1	0.052
OO2DB	Refinement	142	464	85.2%	6	0.042
RSS2ATOM	Refinement	88	154	84%	6	0.068
Tree2Graph	Refinement	15	37	100%	1	0.066
uml2xsd	Migration	17	44	100%	1	0.058
MDDTIF	Refinement	145	377	95.8%	26	0.179
Argouml2ecore	Migration	96	321	79%	13	0.135
StateElimination	Refactoring	313	1062	49.5%	7	0.022
TTC Live Case 2017	Refinement	206	573	79%	6	0.029
uml2Simulink	Refinement	148	477	77%	11	0.074
Average		116.8	348.46	80.5%	8.46	0.072

Table 6. Results summary for ETL

From Table 6 we have that complexity/LOC for ETL is 2.9, indicating a greater semantic density in ETL specifications than for ATL. The rate of flaws per semantic element is 0.024.

3.3 QVT-R

For QVT-R transformations the *CBR* and *UEX* measures are of particular interest, since QVT-R rules (termed 'relations') may be interdependent in several different ways: a rule may refer to another in its *when* or *where* clause, and may have a recursive dependency upon itself, and may override another rule. *UEX* is taken in the worst case as $\frac{n*(n-1)}{2}$ where n is the number of concrete top-level rules in a transformation. A special feature of QVT-R is that relations may define a large number of auxiliary variables to transfer data from one relation domain to another, or to transfer data between relations. This may result in high *EPL* values even for very small transformations. This can cause problems in understanding the relations because the meaning and role of each variable needs to be understood.

The OCL syntax used in QVT-R differs from that of the other MT languages. We evaluate complexity directly on this syntax, rather than upon its OCL translation. Thus an object specification

```
obj : E1 { att = var, rel = obj2 : E2{} }
```

has complexity 11, versus 19 for its conventional OCL equivalent expression:

$$obj : E1 \ and \ obj.att = var \ and \ obj2 : E2 \ and \ obj.rel = obj2$$

We have selected published examples of QVT-R specifications from the ModelMorf repository, from the QVT-R standard, and from published papers [23]. Table 7 gives the measures for the selected QVT-R cases.

Transformation	*ETS* (rs, os)	*ENR*	*ENO*	*ERS*	*EHS*	*EPL*	*EFO*	*CBR*	*DC*	*UEX*
HierarchicalStateMachine2 FlatStateMachine	85 (79, 6)	3	1	0	0	1	0	3(0)	0	3
AbstractToConcrete	47 (47,0)	1	0	0	0	0	0	0(0)	0	0
ClassModelToClassModel	85 (85,0)	3	0	0	0	0	0	4(1)	0	1
DNF	396 (396,0)	9	0	4	0	4	0	10(4)	3	6
DNF_bbox	263 (263,0)	5	0	4	0	5	0	4(0)	3	6
SeqToStm	104 (104,0)	4	0	0	0	1	0	4(0)	0	6
seqtostmct	149 (149,0)	5	0	0	0	0	0	6(3)	0	0
UmlToRdbms	238 (226,12)	7	1	1	0	1	0	10(3)	0	3
UmlToRel	98 (65,33)	2	2	0	0	0	0	3(0)	0	1
RelToCore	2038 (1937, 101)	50	5	11	0	13	5	141(7)	3	15
Bpmn2UseCase	522 (522,0)	23	0	0	0	0	0	12(0)	4	55
hsm2nhdm (recursion)	48 (48,0)	5	0	0	0	0	0	5(2)	0	3

Table 7. Technical debt measures for QVT-R

Table 8 gives a summary of the technical debt of these cases. The same computation of number of flaws is used as for ATL. There are 0.023 flaws/LOC and 0.011 flaws per semantic element, figures intermediate between ATL and ETL. There are 2.09 semantic elements/LOC, a density figure again intermediate between ATL and ETL.

3.4 UML-RSDS

For UML-RSDS transformations we consider three substantial case studies: two parts of the UML2C code generator [20] and the class diagram modulariser *cra* from [17]. We also consider a small specification of a bidirectional transformation (bx), the *family2person* transformation and its inverse [21], the correlation calculator *calc*, computationally complex financial applications *CDO*, Monte-Carlo simulation [18] and a Nelson-Siegal yield-curve estimator [22], and the Transformation Tool Contest refactoring case solutions *pn2sc* [12] and *movies* [13]. The

Transformation	*Category*	*LOC*	$c(\tau)$	*% in rules*	*# flaws*	*flaws/LOC*
HSM2FlatSM	Abstraction	85	137	93%	1	0.011
AbstractToConcrete	Refactoring	47	57	100%	0	0
ClassModelToClassModel	Migration	85	85	100%	2	0.023
DNF	Refactoring	396	665	100%	16	0.04
DNF_bbox	Refactoring	263	470	100%	12	0.045
SeqToStm	Refinement	104	175	100%	1	0.009
seqtostmct	Refinement	149	162	100%	4	0.027
UmlToRdbms	Refinement	238	314	95%	6	0.025
UmlToRel	Refinement	98	75	95%	0	0
RelToCore	Refinement	2038	5415	95%	43	0.021
Bpmn2UseCase	Migration	522	877	100%	7	0.013
hsm2nhdm (recursion)	Abstraction	48	105	100%	2	0.041
Average		339.4	711.25	96%	7.83	0.023

Table 8. Results summary for QVT-R

transformation source files are available at https://nms.kcl.ac.uk/kevin.lano/uml2web/zoo. In total there are 36 individual transformations and 10 transformation systems. Table 9 shows the measures for these transformations and their individual subtransformations. *UEX* is not shown because this is always 0 in UML-RSDS transformations. *EPL* and *ERS* are omitted as they are 0 for all the considered cases. For *ETS*, *ERS* and *EHS* we use $c()$ measures and thresholds of 1000 for transformations and 100 for rules and operations.

Unlike ATL, ETL and QVT-R, UML-RSDS rules cannot call other rules, but only operations of the transformation or of metamodel classes. This simplifies the call graph within the transformation, however at the operation level self-recursive and mutually-recursive calling relations can exist.

Table 10 summarises the results for UML-RSDS. We estimated LOC by printing the specification files and counting lines of operation and use case code, omitting class, generalisation and association declarations.

It can be noted that the $c(\tau)$ measure is around 2.76 times the LOC, a similar level of semantic density to ETL.

An interesting aspect of the results is the balance of functionality between helpers and rules. Excessive use of helpers produces transformations which are akin to programs in a functional programming language. In the largest transformations ($uml2Ca$, $uml2Cb$) there is an imbalance of functionality towards helpers, whilst smaller transformations such as *movies* are more balanced.

4 Analysis

We consider the results for each language with respect to the research questions. For ATL, for **RQ1**, all of the 19 individual transformations had flaws (100%), and 8 of 8 transformation systems contained transformations with flaws (100%).

Transformation	*ETS* (rs, os)	*ENR*	*ENO*	*EHS*	*EFO*	*CC*	*CBR*	*DC*
uml2Ca	1272 (884, 388)	51	31	0	0	0	37(8)	11
(i) *types2C*	177 (177,0)	11	0	0	0	0	0(0)	4
(ii) *program2C*	313 (274, 39)	6	2	0	0	0	3(1)	3
(iii) *printcode*	782 (433, 349)	34	29	0	0	0	34(7)	4
uml2Cb	5621 (904, 4717)	43	124	13	5	32	206(24)	39
(i) *exp2C*	4036 (107, 3929)	8	94	13	5	30	171(18)	38
(ii) *printcode*	1585 (797, 788)	35	30	0	0	2	35(6)	1
cra	1360 (438, 922)	34	56	0	0	6	85(0)	0
(i) *createClasses*	89 (89, 0)	5	0	0	0	0	0(0)	0
(ii) *refactor*	49 (49, 0)	2	0	0	0	0	0(0)	0
(iii) *cleanup*	11 (11, 0)	2	0	0	0	0	0(0)	0
(iv) *measures*	206 (32, 174)	4	6	0	0	2	11(0)	0
(v) *evolve*	374 (50, 324)	4	21	0	0	2	29(0)	0
(vi) *nextgeneration*	293 (82, 211)	10	14	0	0	2	23(0)	0
(vii) *initialise*	286 (84, 202)	5	14	0	0	0	21(0)	0
(viii) *postprocess*	18 (18, 0)	1	0	0	0	0	0(0)	0
(ix) *createClasses*1	34 (23, 11)	1	1	0	0	0	1(0)	0
family2person	45 (45, 0)	3	0	0	0	0	0(0)	2
person2family	113 (91, 22)	5	2	0	0	0	4(0)	1
calc	83 (83, 0)	4	0	0	0	0	0(0)	0
movies	432 (174, 258)	11	8	0	0	2	10(0)	1
(i) *task2*	28 (28, 0)	1	0	0	0	0	0(0)	0
(ii) *task3*	8 (8, 0)	0	0	0	0	0	0(0)	0
(iii) *exttask1*	23 (22, 1)	2	1	0	0	0	2(0)	0
(iv) *exttask2*	39 (39, 0)	2	0	0	0	0	0(0)	1
(v) *exttask3*	8 (8, 0)	1	0	0	0	0	0(0)	0
(vi) *exttask4*	27 (26, 1)	2	1	0	0	0	2(0)	0
(vii) *task1*	263 (7, 256)	1	6	0	0	2	6(0)	0
(viii) *couple2clique*	11 (11, 0)	1	0	0	0	0	0(0)	0
(ix) *nextcliques*	25 (25, 0)	1	0	0	0	0	0(0)	0
Monte-Carlo simulation	90 (61, 29)	6	3	0	0	0	3(0)	0
Nelson-Seigal	1219 (817, 402)	26	24	0	0	0	51(0)	13
(i) *evolve*	202 (68, 134)	5	7	0	0	0	8(0)	0
(ii) *nextgeneration*	134 (84, 50)	9	3	0	0	0	5(0)	1
(iii) *initialise*	768 (591, 177)	10	11	0	0	0	34(0)	12
(iv) *test*	115 (74, 41)	2	3	0	0	0	4(0)	0
CDO	182 (31, 151)	4	9	0	0	0	11(2)	0
(i) *calculateRisk*	170 (19, 151)	3	9	0	0	0	11(2)	0
(ii) *deriveSectorLoss*	12 (12, 0)	1	0	0	0	0	0(0)	0
PetriNet to Statemachine	174 (174, 0)	7	0	0	0	0	0(0)	0
(i) *initialise*	23 (23, 0)	3	0	0	0	0	0(0)	0
(ii) *pn2sc*	120 (120, 0)	2	0	0	0	0	0(0)	0
(iii) *cleanup*	31 (31, 0)	2	0	0	0	0	0(0)	0

Table 9. Technical debt measures for UML-RSDS

Transformation	*Category*	*LOC*	$c(\tau)$	*% in rules*	*# flaws*	*flaws/LOC*
uml2Ca	Code generation	874	1272	69%	22	0.025
uml2Cb	Code generation	1576	5621	16%	119	0.075
cra	Refactoring	490	1360	32%	12	0.024
f2p/p2f	Bidirectional	58	158	86%	3	0.052
calc	Analysis	15	83	100%	0	0
movies	Analysis	156	432	40%	3	0.019
Monte-Carlo sim	Analysis	51	90	68%	0	0
Nelson-Seigal	Refinement	458	1219	67%	15	0.032
CDO	Analysis	94	182	17%	2	0.02
PetriNet to SM	Refactoring	66	174	100%	0	0
Average		383.8	1059.1	34.9%	17.6	0.0458

Table 10. Results summary for UML-RSDS

For **RQ2**, the most common flaws were DC (15/19), CBR (13/19), UEX (10/19), ENR (7/19) and ENO (5/19).

A particular issue in ATL is the use of *resolveTemp* expressions in rules to look up target model elements produced by another rule, during transformation processing. This is considered a semantic complexity factor in [2] because it introduces a syntactic and semantic dependency of the rule calling *resolveTemp* upon the rule identified by the call. We include the rule-to-rule dependencies induced by *resolveTemp* in the CBR measure. Even without this mechanism, other forms of coupling between rules and operations are already a significant quality issue for ATL based on the considered cases.

For ETL, the critical factor in the considered transformations is the implicit *CBR* due to usage of *equivalent* and related operators. For **RQ1**, 19 of the 24 individual transformations contained flaws (79%), and all of the 13 transformation systems contained transformations with flaws (100%). For **RQ2** the most common flaws were *CBR* (18/24), *EFO* (7/24) and *DC* and *UEX* (both 5/24). Excessive size of rules/helpers or transformations was not a significant problem.

For QVT-R, for **RQ1**, out of 12 transformations, 10 had flaws (83%). For **RQ2**, *EPL* and *CBR* both occur in 6 of 12 transformations, whilst *DC* and *ERS* occur in 4. High values of *EPL* arise because of the use of many local variables within QVT-R relations, to facilitate bidirectional use of the relations. *CBR* flaws arise from the unstructured nature of QVT-R transformations in which rules may be closely inter-dependent. In the largest transformation, *relToCore*, there is informal stratification of the transformation into groups of rules, but this would be clearer if the transformation were explicitly decomposed into client and supplier sub-transformations.

For UML-RSDS, for **RQ1**, out of 36 transformations, 16 had some flaws (44%), whilst 7 of 10 transformation systems contained some transformations with flaws (70%). The *uml2Cb* case somewhat distorts the flaw density data: without this case the flaws per LOC would be the same as for QVT-R.

For **RQ2**, excessive *CBR* occurs in 9 transformations (5 involving cycles and 5 excessive numbers of dependencies). *DC* also occurs in 9 cases. *ENO* occurs in 7 cases. *CC* occurs in 6 cases. In all cases, the coupling issues concern complex dependencies between helpers, rather than between rules. The prevalence of *CBR* and *ENO* flaws suggest overuse of helpers/operations. Poor structure and high numbers of flaws were apparent in the largest transformations. Based on Table 10 the correlation of LOC with number of flaws is *0.873*, this is significant at the 5% level using the t-test with 8 degrees of freedom, indicating a strong positive effect of size on the number of flaws, whilst the correlation of number of flaws with the percentage of code in transformation rules is *-0.543* (not statistically significant), indicating that the number of flaws may increase as the proportion of code in rules decreases.

For **RQ3**, Table 11 summarises the different prevalence of TD types in different MT languages, counting the number of transformations which have flaws of each kind. Unusual patterns of TD are emphasised.

TD category	*ATL*	*ETL*	*QVT-R*	*UML-RSDS*	*Overall*
CBR	13/19	18/24	6/12	9/36	46/91
DC	15/19	5/24	4/12	9/36	33/91
UEX	10/19	5/24	2/12	0/36	17/91
ENR	7/19	0/24	2/12	3/36	12/91
ENO	5/19	0/24	0/12	*7/36*	12/91
ERS	5/19	2/24	4/12	0/36	11/91
EFO	0/19	7/24	1/12	1/36	9/91
EPL	0/19	2/24	*6/12*	0/36	8/91
ETS	4/19	0/24	2/12	2/36	8/91
CC	1/19	0/24	0/12	*6/36*	7/91
EHS	1/19	2/24	0/12	1/36	4/91

Table 11. Technical debt prevalence in different MT languages

In summary, it seems that excessive *CBR* and *DC* are the most significant design flaws which arise across all MT languages, although there are significant variations in the kinds of TD problem between different languages. These findings suggest that an important factor in understanding and maintaining model transformations are the dependencies between rules. In particular, rules can be inter-related by mechanisms which lookup or implicitly enact source-target bindings. In ATL, an assignment of objects $g \leftarrow s.f$ cannot be understood without referring to the source metamodel (to identify the type of $s.f$) and in addition, the specification reader then needs to examine any concrete rule in the transformation that accepts this input type, to identify where and how the target object corresponding to $s.f$ has been created from $s.f$. However the semantics of ATL execution ensures that this rule-to-rule dependency can be statically determined and understood. In ETL the assignment $t.g ::= s.f$ may in addition lead to (possibly recursive) invocations of rules to convert $s.f$ to a target element to assign to $t.g$, the rules executed may depend on the runtime values of the data of $s.f$.

In contrast, QVT-R specifications explicitly state the rules responsible for source-target bindings that are used in such assignments: rules either already executed (the *when* clause) or to be executed (the *where* clause). Rules can however be involved in recursive loops of rule dependencies.

In UML-RSDS, an assignment $t.g = \mathit{TEnt}[f.\mathit{sId}]$ makes explicit the source-target correspondence of f and $\mathit{TEnt}[f.\mathit{sId}]$. This correspondence must have been established by a preceding rule in the sequential control flow.

CBR could be reduced by the stratification and modularisation of transformations into smaller units. Currently MT languages offer such *external* composition [28] of transformations by the sequencing of individual transformations: a facility heavily used in the UML-RSDS examples in particular. However it seems what is needed is a modularisation mechanism to support a hierarchical client-supplier relationship between transformations, with the internal details of the supplier module independent of its clients. This would enable, for example, a transformation mapping OCL expressions to be called as a 'black box' from a transformation mapping UML activities. The combination of these two transformation processes into the single UML-RSDS *exp2C* case is a significant cause of flaws in *uml2Cb*.

Table 12 shows the overall figures for LOC, c, and flaws, for each language.

Language	LOC	c	c/LOC	Flaws	Flaws/LOC	Flaws/c
ATL	6029	10311	1.71	96	0.018	0.009
ETL	1519	4530	2.98	110	0.072	0.024
QVT-R	4073	8535	2.09	94	0.023	0.011
UML-RSDS	3838	10591	2.76	176	0.046	0.017
Overall	15459	33967	2.19	476	0.031	0.014

Table 12. Overall size and TD results

We can also compare the levels of TD in different categories of transformation, across languages. Table 13 shows the TD frequency for the main categories of transformations in our survey. Although the sample numbers are too small for statistical significance, the difference in flaw levels between the main categories is in accord with expectations that more complex MT tasks such as refactoring will result in transformations with higher numbers of flaws compared to simpler tasks such as migration.

For **RQ4**, TD densities in developer-coded Eclipse projects have been measured in [8], with values ranging from 0.005 to 0.04 flaws per LOC, with an average around 0.015. We also evaluated manually coded versions of a UML to C++ translator (18,100 lines of Java), and of the CDO case study (200 lines of C++) using the PMD code size library (https://pmd.github.io). These had similar levels of TD (0.009/LOC and 0.021/LOC, respectively) to the MT language cases. The TD levels of ETL and UML-RSDS are high in comparison with

Category	LOC	Flaws	Flaws/LOC
Code generation	2613	143	0.055
Bidirectional	58	3	0.052
Refinement	4280	133	0.031
Refactoring	1575	47	0.029
Migration	4222	103	0.024
Abstraction	1128	22	0.019
Analysis	316	5	0.016
Semantic map	1267	20	0.016

Table 13. TD for MT categories

these code TD results, whilst ATL and QVT-R exhibit TD levels more typical of executable code.

5 Reducing technical debt in MT

The technical debt measures help to identify problem areas in particular transformations, such as rules which exceed the thresholds in terms of size, interdependencies, etc. Such rules can then be refactored, as can identified cases of large duplicated expressions.

In general, if a rule r exceeds the size threshold of *ERS*, $c(r) > 100$, the specifier should identify a subpart P of r which can be factored out as a separate called rule or operation $r1$. The complexity of $r1$ is $c(P)$, whilst $c(r)$ is reduced by $c(P) - x$ where x is the complexity of a call to $r1$. $c(P)$ should be significantly larger than x, and sufficiently so to ensure that $c(r) - c(P) + x < 100$, whilst $c(P) < 100$. The same refactoring applies to called rules and operations that exceed size thresholds (*EHS*).

The operations *mapAttributeExpression* and *mapRoleExpression* from *uml2Cb* are examples of this situation: both have complexities over 200 (the worst examples in this transformation). They are also the source of some of the largest cases of duplicate code in the transformation. The operations can each be refactored into four new subordinate operations to handle the separate cases of attribute/role basic expressions.

Such a refactoring can also reduce *DC* and *CC* measures, but does not necessarily reduce the overall size of a transformation, indeed it may increase it, and it may increase the measures *ENR* or *ENO* of number of rules/operations, and the number of dependencies *CBR* in the call graph.

To reduce the overall size of a large transformation, it can be factored into a sequential composition of smaller transformations (the Transformation Chain MT pattern [19]). Decomposition into sequenced subtransformations is a technique for introducing transformation 'phases' [7], and can reduce the *UEX* metric and *CBR*. More powerful composition mechanisms, such as using one transformation as a supplier module to another, may need to be introduced into MT languages.

Cyclic dependencies in *CBR* can be caused by recursively-defined operations based upon self-associations in the source metamodel, as with *uml2Ca* and *uml2Cb*. These can be restructured to avoid recursion, by instead pre-computing the transitive closure of the self-association.

6 Threats to validity

The conclusions we have drawn may be challenged on the basis that (a) the measures chosen are not appropriate for evaluating TD; (b) the selection of transformation cases was unrepresentative; (c) the basis of TD measurement of different MT languages are not equivalent.

Regarding (a), we have adopted established TD measures which have been used extensively for TD evaluation of programs. In practice, the UML-RSDS transformations considered have all been the subject of maintenance activities, and the problem areas located by measures of excessive rule/operation size, call-graph complexity and code duplication accord with our experience of the most time-consuming parts for maintenance. Excessive size and calling complexity hinders location of elements to modify, and understanding of the impact of modifications. Duplication leads to duplication of modifications. These factors should apply also to other MT languages. We have used 500 LOC as a threshold for transformation size, and 50 LOC as a threshold for rule/operation size. This is partly justified by the fact that overall the ratio of complexity to LOC is close to 2, and thus the 50/500 LOC limits correspond, on average, to the 100/1000 limits for syntactic complexity. In addition, out of 74 cases where both transformation LOC and $c(\tau)$ were available, in 69 cases (93%) the thresholds were in agreement: both $c(\tau) > 1000$ and $LOC > 500$ in 9 cases, or both $c(\tau) \leq 1000$ and $LOC \leq 500$ in 60 cases. Two cases were over 500 LOC but below 1000 $c(\tau)$ whilst 3 had the converse.

Regarding (b), we have considered public repositories of cases and published examples of MT specifications for each language, and the selection of cases has been on the same basis for each language. For each language, we have endeavoured to obtain a wide range of transformation examples, spanning in size from small cases to the largest cases available, and across the range of all available categories of transformation. It can be noted that the ETL cases are significantly smaller (average complexity size 348) than the ATL, QVT-R or UML-RSDS cases (average sizes 1289, 711 and 1059). There are few large publicly-available ETL cases, which restricted our choice for analysis.

Regarding (c), some distortion is introduced by the analysis of cases where one MT language feature is used to express another concept in the source specification. For example, in the KM32DOT ATL transformation, the first 9 helper operations *DiagramType*(), *Mode*(), etc are used to represent the parameters of the transformation. These operations are also represented as helper operations in the abstract syntax representation of KM32DOT, which is semantically correct, but not consistent with the *intent* of the original ATL helpers. Such cases would require manual correction in the analysis, but we consider that it is preferable to analyse the transformations on the basis of their actual text, not on the basis of how the specifier intended the text to be interpreted (since this knowledge may not be available in some cases, leading to inconsistency in the analysis).

There is a close abstract syntax correspondence between ATL, QVT-R, ETL and UML-RSDS, with similar concepts of top-level rules, called (non-top) rules, and auxiliary operations/helpers. However there are significant differences in the semantics of the languages, and mappings from the source text to some common semantic representation of the languages, as described in [15], would substantially alter the measures of the transformation specifications. Thus we have performed measurement on the abstract syntax, using variations on measures (eg., for *CBR* and *UEX*) to take account of different language semantics.

7 Related work

One of the first works to consider metrics for MT was [10]. They define measures for the size and complexity of QVT-R transformations, including lines of code, number of relations (corresponding to number of rules), and specific measures for the size and inter-relationship of QVT-R rules. Their analysis is limited to QVT-R and does not consider clone detection or detailed analysis of the rule dependency graph. They evaluated one large (auto-generated) QVT-R transformation and three moderate/small transformations. Undefined execution order between rules is a significant problem in the large transformation. In [1], measures of ATL and QVT-R and QVT-O are computed for versions of two transformations in each language. Since the transformations are small-scale, no design flaws are detected from the measures, although these help to illustrate differences in the structure of the three versions. As expected, the size (complexity) of the QVT-O version is larger than for QVT-R or ATL due to the more procedural nature of QVT-O. In [2], seven ATL transformations are evaluated by metrics and by expert analysis, in order to identify correlations between metric values and expert evaluation of quality characteristics. Wimmer et al [30] use quality measures to evaluate the effect of MT refactorings. They adopt ERS, DC and EFO as quality criteria for ATL transformations. They also consider that OCL expressions with complexity greater than 10 are a potential quality flaw.

Clone detection in transformations is considered by [26], and they evaluate alternative tools for clone detection in graph transformations. Duplicate sequences of actions in use cases are detected in the approach of [25], and their technique could potentially be applied to transformations abstracted as use cases.

There have been several studies comparing different MT languages. The annual Transformation Tool Contest compares different languages on a variety of case studies. In [11] we compare UML-RSDS, QVT-R and ATL on a refactoring case using several quality measures. It was found that the call-graph complexity of the ATL and QVT-R solutions were significantly higher than for the UML-RSDS solution, and the ratio of complexity to size for ATL and QVT-R were 1.77 and 2.13, similar to those we have found in this paper.

Conclusion

We have shown that technical debt can be evaluated for different MT languages. We have evaluated 91 transformations in four transformation languages, and identified significant differences between these in their frequency and kind of TD. The identification of design flaws can help MT specifiers to improve their transformations and to prioritise refactoring or other quality improvement work on their transformations.

References

1. M. van Amstel, S. Bosems, I. Kurtev, L. Pires, *Performance in model transformations: experiments with ATL and QVT*, ICMT 2011, LNCS 6707, pp. 198–212, 2011.
2. M. van Amstel, M. van den Brand, *Using metrics for assessing the quality of ATL model transformations*, MtATL 2011.
3. T. Arendt, G. Taentzer, *UML model smells and model refactorings in early software development phases*, Technical report FB 12, Philipps Universitat, Marburg, 2010.
4. V. Basili, *Software modeling and measurement: the goal/question/metric paradigm*, 1992.
5. J. Cabot, E. Teniente, *A metric for measuring the complexity of OCL expressions*, Workshop on model size metrics, MODELS '06, 2006.
6. A. Correa, C. Werner, *Refactoring OCL specifications*, SoSyM 6: 113–138, 2007.
7. J. Cuadrado, J. Molina, *Modularisation of model transformations through a phasing mechanism*, SoSyM vol. 8, no. 3, 2009, pp. 325–345.
8. X. He, P. Avgeriou, P. Liang, Z. Li, *Technical debt in MDE: A case study on GMF/EMF-based projects*, MODELS 2016.
9. IEC/ISO, *9126 Software engineering – Product quality – Part 1: Quality model*, 2001.
10. L. Kapova, T. Goldschmidt, S. Becker, J. Henss, *Evaluating maintainability with code metrics for model-to-model transformations*, Research into Practice – Reality and Gaps, Springer, 2010.
11. S. Kolahdouz-Rahimi et al., *Evaluation of MT approaches for model refactoring*, Sci. Comp. Prog., vol. 85, 2014.
12. K. Lano, S. Kolahdouz-Rahimi, K. Maroukian, *Solving the Petri-Nets to Statecharts Transformation Case with UML-RSDS*, TTC 2013, EPTCS, 2013.
13. K. Lano, S. Yassipour-Tehrani, *Solving the TTC 2014 Movie Database Case with UML-RSDS*, TTC 2014.
14. K. Lano, S. Kolahdouz-Rahimi, *Model-transformation Design Patterns*, IEEE Transactions in Software Engineering, vol 40, 2014.
15. K. Lano, S. Kolahdouz-Rahimi, S. Yassipour-Tehrani, *Analysis of hybrid MT language specifications*, FSEN 2015.
16. K. Lano, S. Kolahdouz-Rahimi, S. Yassipour-Tehrani, *Model transformation semantic analysis by transformation*, VOLT 2015.
17. K. Lano, S. Kolahdouz-Rahimi, S. Yassipour-Tehrani, *Solving the CRA case using UML-RSDS*, TTC 2016.
18. K. Lano, *Agile model-based development using UML-RSDS*, CRC Press, 2016.
19. K. Lano, S. Kolahdouz-Rahimi, S. Yassipour-Tehrani, M. Sharbaf, *A survey of model transformation design pattern usage*, ICMT 2017.

20. K. Lano et al., *Translating from UML-RSDS OCL to ANSI C*, OCL 2017.
21. Kevin Lano, Shekoufeh Kolahdouz-Rahimi, *Families to Persons Case with UML-RSDS*, TTC 2017.
22. K. Lano, H. Haughton, et al. *Agile model-driven engineering of financial applications*, FlexMDE, MODELS 2017.
23. N. Macedo, A. Cunha, *Least-change bidirectional model transformation with QVT-R and ATL*, SoSyM (2016) 15: 783–810.
24. R. Marinescu, *Assessing technical debt by identifying design flaws in software systems*, IBM Journal of Research and Development, 56(5), 2012.
25. A. Rago, C. Marcos, J. Diaz-Pace, *Identifying duplicate functionality in textual use cases by aligning semantic actions*, SoSym vol. 15, no. 2, 2016.
26. D. Struber, J. Ploger, V. Acretoaie, *Clone detection for graph-based MT languages*, ICMT 2016.
27. S. Yassipour-Tehrani, S. Zschaler, K. Lano, *Requirements Engineering in Model-transformation Development: an interview-based study*, ICMT 2016.
28. D. Wagelaar, *Composition techniques for rule-based MT languages*, ICMT 2008.
29. G. B. Weatherill, *Elementary statistical methods*, Chapman and Hall, 1978.
30. M. Wimmer, et al., *A Catalogue of Refactorings for model-to-model transformations*, Journal of Object Technology, vol. 11, no. 2, 2012.