Integrating Technical Debt into MDE

Fáber D. Giraldo[1,2], Sergio España[2], Manuel A. Pineda[1], William J. Giraldo[1] and Oscar Pastor[2]

[1] System and Computer Engineering, University of Quindío, Colombia
email: {fdgiraldo, mapineda, wjgiraldo}@uniquindio.edu.co
[2] PROS Research Centre, Universitat Politècnica de València, Spain
email: {fdgiraldo,sergio.espana,opastor}@pros.upv.es

Abstract. The main goal of this work is to evaluate the feasibility to calculate the technical debt (a traditional software quality approach) in a model-driven context through the same tools used by software developers at work. The *SonarQube* tool was used, so that the quality check was performed directly on projects created with Eclipse Modeling Framework (EMF) instead of traditionals source code projects. In this work, XML was used as the model specification language to verify in SonarQube due to the creation of EMF metamodels in XMI (XML Metadata Interchange) and that SonarQube offers a plugin to assess the XML language. After this, our work focused on the definition of model rules as an XSD schema (XML Schema Definition) and the integration between EMF-SonarQube in order that these metrics were directly validated by SonarQube; and subsequently, this tool determined the technical debt that the analyzed EMF models could contain.

Keywords: Model-driven engineering, technical debt, EMF, SonarQube.

1 Introduction

Two representative trends for the software development industry that appeared in the nineties were the *model-driven* initiative and the *technical debt* metaphor. Both trends promote software quality each in its own way: high abstract levels (models) and software process management (technical debt). However, despite the wide exposition of these trends in the literature, there are not more indications about the combination of them into software development scenarios; each initiative is implemented in a separated way.

In traditional software development projects (those involving manual programming), technical debt is mainly focused in quality assurance processes over source code and related services (e.g. common quality metrics are defined over source code). However, model-driven engineering (MDE) promotes for modelling instead of programming [1]. A review of the literature reveals that there is currently no application of the technical debt concept to environments outside the traditional software development. There exist approaches to the measurement of model quality [7][8][9][10], but these do not include technical debt calculus.

Therefore, we claim that dealing with technical debt in MDE projects is an open problem.

Two issues pose challenges to the inclusion of technical debt into MDE. *(i)* Different authors provide conflicting conceptions of quality in model management within MDE environments[5]. *(ii)* The MDE literature often neglects techniques for source code analysis and quality control[3]. Therefore, in model-driven developments it is difficult to perform an analysis of the state of the project that is important for technical debt management: establishing what has been done, what remains to be done, how much work has been left undone. Also, other specific issues that belong to model theory such as: number of elements in the metamodel, coverage for the views, complexity of the models, the relationship between the abstract syntax and the concrete syntax of a language, quantity of OCL verification code, among others, contribute to increase the technical debt in model-driven projects.

The main contributions of this paper are the following: *(i)* A demonstration of a integration between model-driven and technical debt tools for supporting a technical debt calculus process performed over conceptual models. *(ii)* The operationalization of a recognized framework for evaluating models.

2 Implementation of a technical debt plugin for EMF

We implemented an Eclipse plugin for integrating the EMF enviroment with SonarQube; so that, results of the technical debt can be shown directly on the Eclipse work area instead of changing the context and opening a browser with the SonarQube report. We used configuration options belonging to EMF *XMIResource* objects to export the XMI file as an XML without the specific XMI information tags.

2.1 Definition of an XSD for SonarQube

One of the most critical issues in a technical debt program is the definition of metrics or procedures for deducting technical debt calculations; in works like [4][6] it is highlighted the absence of technical debt values (established and accepted), and features such as the kinds of technical debt. Most of the technical debt calculation works are focused on software projects without an applied model-driven approach; some similar works report the use of high level artifacts as software architectures[12], but they are not model-driven oriented. Emerging frameworks for defining and managing technical debt [13] are appearing, but they focus on specific tasks of the software development (not all the process itself).

[3] Neglecting the code would seem sensible, since MDE advocates that the model _is_ the code [2]. However, few MDE tools provide full code generation and manual additions of code and tweakings are often necessary.

From one technical perspective, the SonarQube tool demands an XSD (XML Scheme Document) configuration file that contains the specific rules for validating the code; or in this case, a model. Without this file, the model could be evaluated like a source code by default. In order to define these rules, we chose one of the most popular proposals for validating models (*Physics of notations* of Moody [11]) due to its relative easiness to implement some of its postulates in terms of XSD sentences.

In the case of this work, visual notation was taken as the textual information managed by XMI entities from EMF models (text are perceptual elements too), focusing that each item meets syntactic rules to display each information field regardless about what is recorded as a result of the EMF model validation. The analysis does not consider the semantic meaning of the model elements to be analyzed.

The operationalization of Moody principles over the XSD file posteriorly loaded in SonarQube was defined as follows:

- *Visual syntax - perceptual configuration*: in the XSD file, it is ensured that all elements and/or attributes of the modelled elements are defined according to the appropriate type (the consistence between the values of attributes and its associated type is validated).
- *Visual syntax - attention management*: a validation order of the elements is specified by the usage of order indicadors belonging to XML schemes.
- *Semiotic clarity - redundant symbology*: a node in the model can only be checked by an XSD element.
- *Semiotic clarity - overload symbology*: an XSD element type only validates a single model node type.
- *Semiotic clarity - excess symbolism*: a metric to validate that there are no blank items was implemented (for example, we could create several elements of *Person* type, but its data does not appear).
- *Semiotic clarity - symbology deficit*: a validation that indicates the presence of incomplete information was made (e.g., we could have the data of a *Person* but we don't have his/her name or identification number). For this rule, we made constraints with occurrence indicators to each attribute.
- *Perceptual discriminability*: in the XML model, nodes must be organized in a way that they can be differentiated, e.g., one *Project* element does not appear like a *Person* element. This is ensured by reviewing in the XSD that it does not contain elements exactly alike, and in the same order.
- *Semantic transparency*: this was done by putting restrictions on the names of the tags, so that the tags correspond to what they must have, e.g, a *data* label must be of *data* type.
- *Complexity management*: this was done by the *minOccurs* and *maxOccurs* occurrence indicators. With these indicators it is possible to define how many children one node can have.
- *Cognitive integration*: this was done using namespaces in the XSD file, so that it is possible to ensure the structure for the nodes independent from changes in the model design.

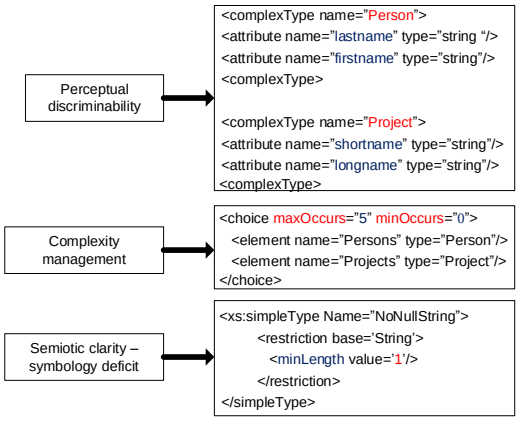

Fig. 1. Mapping between some Moody principles to XSD code.

- *Dual codification*: this was done by measuring the quantity of commented code lines with respect to the XML lines that define the elements of the model.
- *Graphic economy*: we established a limit for different items that can be handled in the XSD, and reporting when different elements are found marking the mistake when these data types are not found in the schema.
- *Cognitive fit*: this was done by creating several XSD files where each one is responsible for reviewing a specific view model.

Figure 1 exposes a portion of the XSD code implemented for some Moody principles.

2.2 Verification of technical debt from EMF models

In order to demonstrate the integration of both tools (EMF-SonarQube), a sample metamodel (Figure 2) was made in EMF[4]. We introduce some errors like capital letters, blank spaces, and others, to evidence abnormalities not covered with model conceptual validation approaches like OCL.

Once the validation option had been chosen (by the SonarQube button or menu), we obtain a report similar to Figure 3. Part *A* indicates the number of lines of code that have been tested, comment lines, and duplicate lines, blocks or files. Also, part *B* of this figure reports the total of errors that contain the project (in this case the EMF model), as well as the technical debt graph (part *C*), which shows the percentage of technical debt, the cost of repair, and the

[4] this metamodel was extracted from a web tutorial about EMF; source: http://tomsondev.bestsolution.at/2009/06/06/galileo-emf-databinding-part-1/

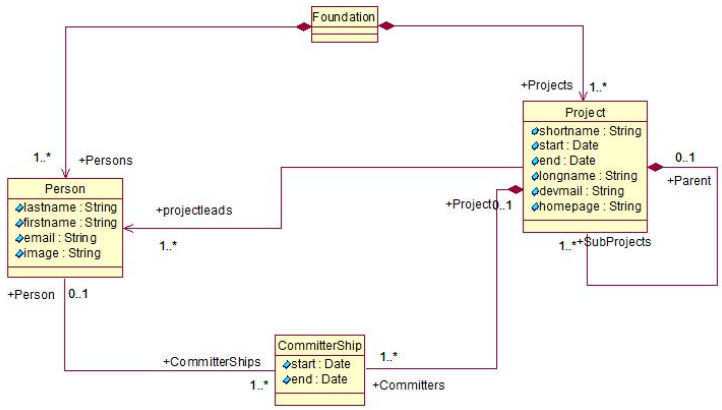

Fig. 2. Sample metamodel implemented over EMF.

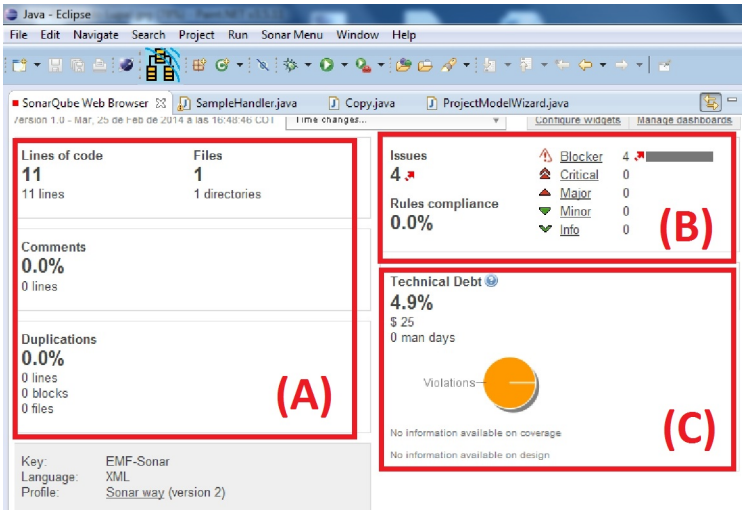

Fig. 3. SonarQube screen report loaded into EMF work area

number of men needed to fix errors per day (this information was not configured for this case).

SonarQube offers an *issues* report where it indicates the number of errors found; and consequently, the error list distributed in order of importance from highest to lowest:

– *Blocker*: they are the most serious errors; they should have the highest priority to review.

– *Critical*: they are design errors which affect quality or performance of the project (model errors can be classified in this category).
– *Major*: although these errors do not affect performance, they require to be fixed for quality concerns.
– *Minor*: they are minor errors that do not affect the operation of the project.
– *Info:* they are reporting errors, not dangerous.

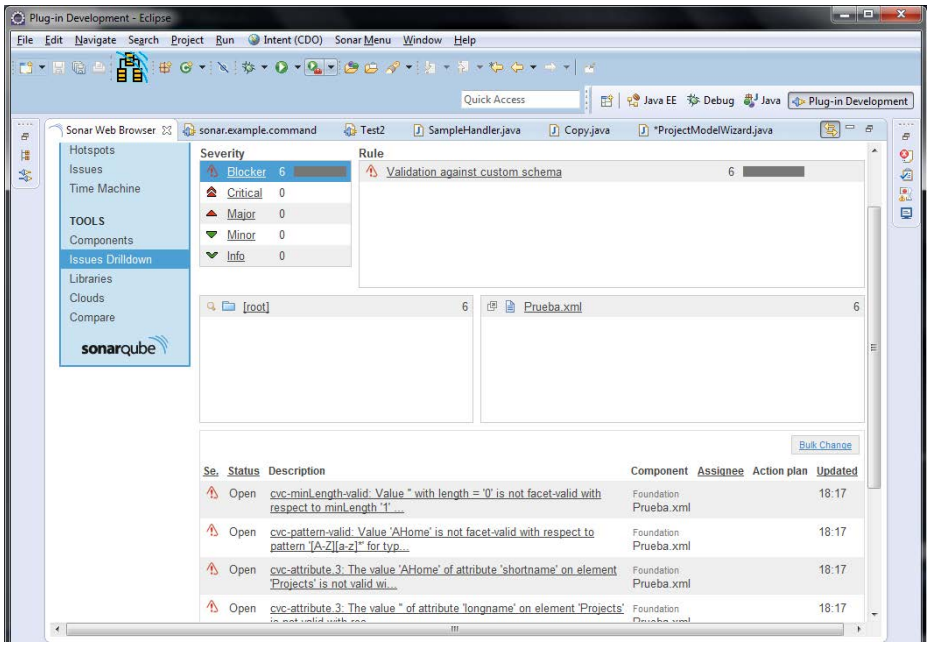

Fig. 4. SonarQube issues report of technical debt in EMF.

Figures 4 and 5 present the reports about technical debt errors detected over the sample model. In the first place, an error category was chosen. For the respective category, the error list associated is show in detail posteriorly. Intentionally, we introduced errors over the XML information of the model to test the respective detection by SonarQube according with the rules defined in the XSD file from the Moody proposal.

Conclusions

In this work we show the technical feasibility to integrate a technical debt tool like SonarQube with a model-driven development enviroment such as the Eclipse modelling framework. We present an example of technical debt validation applied

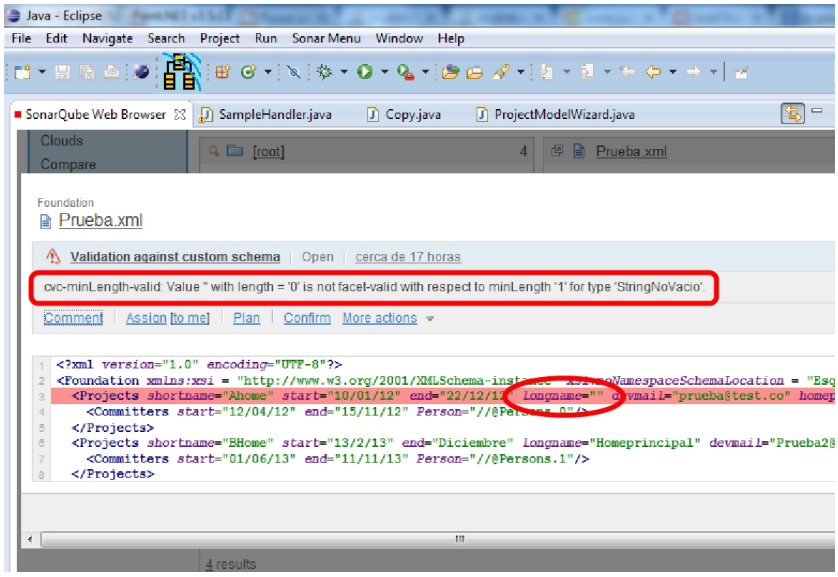

Fig. 5. Example of error detected by SonarQube.

over a sample metamodel implemented for testing purposes. Thereby, we demonstrate the technical feasibility for measuring any artefact used in an model-driven engineering process [3]. However, the main challenge is the definition of the model quality metrics and the operationalization of the model quality frameworks reported in terms of expressions that can generate metrics, and its association with a model-driven development process.

Acknowledgments

F.G, thanks to Colciencias (Colombia) for funding this work through the Colciencias Grant call 512-2010, and the Pan-American Software Quality Institute (PASQI) for the participation and feedback received in the PASQI Workshop 2013. F.G. and M.P. thanks to David Racodon (david.racodon@sonarsource.com) and Nicla Donno (nicla.donno@sonarsource.com) for their suppport with the SQALE plugin for SonarQube. This work has been supported by the Spanish MICINN PROS-Req (TIN2010-19130-C02-02), the Generalitat Valenciana Project ORCA (PROMETEO/2009/015), the European Commission FP7 Project CaaS (611351), and ERDF structural funds.

References

1. Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice.* Synthesis Lectures on Software Engineering. Morgan &

Claypool Publishers, 2012.
2. David W. Embley, Stephen W. Liddle, and Oscar Pastor. Conceptual-model programming: A manifesto. In David W. Embley and Bernhard Thalheim, editors, *Handbook of Conceptual Modeling*, pages 3–16. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-15864-3.
3. Bertoa Manuel F. and Vallecillo Antonio. Quality attributes for software metamodels. In *Proc. of 13th TOOLS workshop on quantitatives approaches in object-oriented software engineering. QAAOSE 2010. July 2nd. Málaga, Spain*, February 2010.
4. D. Falessi, M.A. Shaw, F. Shull, K. Mullen, and M.S. Keymind. Practical considerations, challenges, and requirements of tool-support for managing technical debt. In *Managing Technical Debt (MTD), 2013 4th International Workshop on*, pages 16–19, 2013.
5. Peter Fettke, Constantin Houy, Armella-Lucia Vella, and Peter Loos. Towards the reconstruction and evaluation of conceptual model quality discourses - methodical framework and application in the context of model understandability. In Ilia Bider, Terry A. Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer, and Stanislaw Wrycza, editors, *BMMDS/EMMSAD*, volume 113 of *Lecture Notes in Business Information Processing*, pages 406–421. Springer, 2012. ISBN 978-3-642-31071-3.
6. C. Izurieta, I. Griffith, D. Reimanis, and R. Luhr. On the uncertainty of technical debt measurements. In *Information Science and Applications (ICISA), 2013 International Conference on*, pages 1–4, 2013.
7. John Krogstie. Quality of models. In *Model-Based Development and Evolution of Information Systems*, pages 205–247. Springer London, 2012. ISBN 978-1-4471-2935-6.
8. C.F.J. Lange and M.R.V. Chaudron. Managing Model Quality in UML-Based Software Development. In *Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on*, pages 7–16, 2005. LCCN 0029.
9. Beatriz Marín, Giovanni Giachetti, Oscar Pastor, and Alain Abran. A quality model for conceptual models of mdd environments. *Adv. Soft. Eng.*, 2010:1:1–1:17, January 2010.
10. Parastoo Mohagheghi and Vegard Dehlen. Developing a quality framework for model-driven engineering. In Holger Giese, editor, *Models in Software Engineering*, volume 5002 of *Lecture Notes in Computer Science*, pages 275–286. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-69069-6.
11. Daniel L. Moody. The ´physics´ of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 2009.
12. R.L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas. In search of a metric for managing architectural technical debt. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 91–100, 2012.
13. Carolyn Seaman and Yuepu Guo. Chapter 2 - measuring and monitoring technical debt. volume 82 of *Advances in Computers*, pages 25 – 46. Elsevier, 2011.