Human factors in the adoption of model-driven engineering: an educator's perspective

Jordi Cabot[1,2] and Dimitrios S. Kolovos[3]

[1] ICREA
[2] Internet Interdisciplinary Institute, UOC
jordi.cabot@icrea.cat
[3] Department of Computer Science, University of York, UK
dimitris.kolovos@york.ac.uk

Abstract. This paper complements previous empirical studies on teaching Model-driven Engineering (MDE) by reporting on the authors' attempt at introducing MDE to undergrad students. This is important because: 1) today's students are tomorrow's professionals and industrial adoption depends also on the availability of trained professionals and 2) observing problems in the introduction of MDE in the more controlled environment of a classroom setting allows us to identify additional adoption factors, more at the individual level, to be taken into account after in industrial settings. As we report herein, this attempt was largely unsuccessful. We will analyze what went wrong, what we learned from the process and the implications this has for both future endeavors of introducing MDE in both educational and professional environments, particularly regarding human/socio-technical factors to be considered.

Keywords: Education, Model-driven Engineering, code-generation, empirical

1 Introduction

Model-driven engineering (MDE) has not been adopted by industry as extensively as many expected. Stephen Mellor (creator of the Executable UML [17] concept) is famously quoted saying that *modeling will be commonplace in three years time* with the only *but* being that he has been giving the same prediction since 1985. Many empirical studies have investigated this phenomenon (see [22] for a recent overview), sometimes even with contradictory results and almost always accompanied with heated discussions e.g. [19]. There is a consensus, though, to point out to organizational and managerial reasons and not only technical concerns as key aspects of this limited adoption, as it has happened before with other technologies [8]. Choosing the wrong project for a first MDE pilot, not having enough internal support, the lack of expertise of the development team in MDE technologies are often given as examples, based on developers' interviews and surveys.

An initial failure creates a resistance to try again in the future, further hampering the adoption of MDE in industry. To overcome this situation we believe it is important that this first MDE impression is given in a course part of computer science/engineering degrees where the more controlled environment of a classroom setting can be used to isolate students from the contextual problems professionals have to face and maximize the chances to convince students of the potential of MDE and (conceptual) modeling in general. This would also fix the *lack of expertise* adoption challenge mentioned above.

At least, this was the idea. But (on two instances) such an attempt proved largely unsuccessful. The goal of the paper is to describe how an attempt to introduce MDE to undergraduate students by the first author at École des Mines de Nantes failed and what lessons can be learnt from that. As discussed in the sequel, most factors involve regarding the adoption of MDE as a socio-technical problem where human factors/ergonomy aspects of the interaction with the tools play an important role. This study offers a new perspective in this field by complementing other empirical studies more focused on interviewing practitioners. The lessons learned can be useful for any (not necessarily educational) institution or organization that would like to adopt MDE and also for tool vendors to better understand how they can improve their tools.

The rest of the paper is structured as follows. The next section provides a critical review of related studies, Sections 3 and 4 describe the MDE courses and the feedback students provided on them, Section 5 summarizes the lessons learned from the experience, and Section 6 concludes the paper and discusses future work.

2 Related Work

Many research works have tried to develop insights on how model-driven engineering (and software modeling) is used by practitioners. Some focus on specific languages (mostly on UML [7, 11, 15, 18, 19]), specific phases in the development process (like software maintenance [12, 13]) or individual (types of) companies or techniques [4, 9], while some others aim at getting a more broader picture of why MDE is (not) adopted by looking at a variety of organizations in different domains and categories [14, 22, 23].

Sometimes these papers show inconsistent results and are often received with heated discussions, as happened with one of the latest works [19]. It is therefore important to continue this line of research by complementing existing empirical studies with new observations, particularly in domains / areas that have not been well-covered so far, as it is the case for this paper.

None of these previous works focuses on an experiment in a classroom setting like the one we report here. Using students as subjects in empirical studies is always controversial but in this context we believe it is valuable to analyze how to improve the first impression of people exposed to MDE for the first time since 1) today's students are tomorrow's professionals and industrial adoption depends also on the availability of trained professionals and 2) observing problems in the introduction of MDE in a more controlled environment allows us to identify additional adoption factors, at an individual level, to be taken into account in industrial settings and that could have been missed when interviewing developers in other studies due to the complex environment (organizational, managerial,...) they work in.

The results we discuss below do not replace previous experiences but complement them, contributing to this global effort in improving (or at least understanding) the role of MDE in general in the software engineering community.

3 Introductory MDE Course: Version 1

The first version of the introductory MDE course at École des Mines de Nantes was taught as part of the 2012/13 academic year by the first author and was aimed at providing undergrad students of the engineering degree (CS specialization) with a complete overview of all main MDE aspects using the Eclipse/EMF [21] framework (the standard *de facto* in the community) as base platform to illustrate those concepts with actual MDE tools. The idea was to show students how to build the MDE artifacts (transformations, generation templates and even DSLs) required to integrate MDE in their development process. Using EMF across the course helped save time on the tooling aspects by providing a uniform platform for the whole course. The total number of teaching hours for the course was 45h (this does not include personal time outside class to work on the personal assignments), given by the first author in combination with external lecturers invited to participate in specific sessions based on their expertise. Students had taken previously an introductory course on UML and design patterns.

The course comprised three sections: MDE Foundations (5h), MDE Core technologies (30h), and Methodology and infrastructure (10h). For each section students were asked to complete a small exercise either individually on in small groups of 2-3 students. These exercises always involved the development of a new MDE artifact (a new model-to-model transformation, a new DSL ...) needed to build a MDE process for the needs of a fictitious company, therefore, students were playing the role of MDE developers during the course. To simplify the assignments, in some cases, skeletons or initial versions of the artifacts to develop were provided alongside the requirements. In a couple of cases, assignments had to be simplified, or even the requirement to build a running product had to be dropped, due to the lack of proper documentation from the tools chosen for the course. No good tooling alternative was available that could be used to replace those problematic tools.

This approach was tried for two consecutive years (2012/13 and 2013/14) until we realized and learned the hard way that you cannot ask people to become MDE developers if they are not first convinced and experienced MDE users (which is what most of them will end up being, if anything). This realization was not linked to poor students' marks. In fact marks were not bad at all but it was because they managed to "fill the gaps" in the individual tasks even when they were not really sure what they were doing (or why they were doing it). By talking to them one could see they had no real understanding or coherent view of MDE as a whole and just went in surviving mode from one tool and assignment to the other until the end of the course. This message was not only our perception as teachers but it was also conveyed by the students themselves to the dean of studies as part of the feedback students are asked to provide about all courses every year.

4 Introductory MDE Course: Version 2

After two largely unsuccessful editions of the course we realized that trying to cover all aspects of building MDE artifacts was too much and there was no real practical reason for that since the chances of students working professionally as language designers or similar are rather small. Moreover, they were not convinced about the benefits of MDE in the first place so their motivation was low as well.

Therefore we decided to shift the focus to a more “MDE as users” perspective. The rationale was that this is a more reasonable goal for a first MDE course and one that, if fulfilled, would achieve our goal of increasing the adoption of MDE in software projects. We would like to train students to be at least open to use MDE in their future professional projects, and thus, a course on basic MDE principles from a user perspective seemed more appropriate and a better way to motivate them.

We still wished to cover in the course all the main concepts and components of MDE but now aiming at helping students to understand what each MDE technique was useful for and how they could add it to their arsenal of software tools (even if this could first imply hiring a MDE language designer to adapt the tools to their own specific context before they can start developing software with them). To avoid the *too many too shallow* problem we decided to devote ample time to a couple of case studies, including a large one where students would need to develop a web-based application using MDE techniques (mainly code generation) with the hope this scenario would be a very illustrative and convincing way to demonstrate the advantages MDE could bring to their daily professional practice particularly in terms of productivity and quality gains.

With this new vision, the reshaped course syllabus evolved as follows (in bold the parts that changed from the previous edition, including changes in content and time distribution)

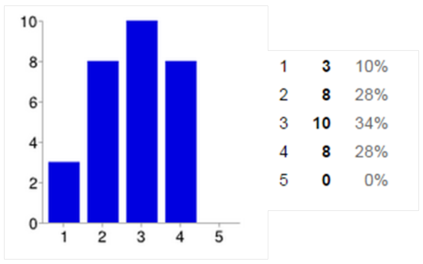
1. MDE Foundations **(4h)**
2. MDE Core technologies **(8h)**
3. Methodology and infrastructure **(3h)**
4. **Case Study 1: Model-driven Reverse Engineering with MoDisco (8h)**
5. **Case Study 2: Model-driven Software Development with WebRatio (22h)**

Note the drastic reduction in the core technologies section. Before, the time devoted to this part included the assignments to develop several MDE artifacts. In the revised syllabus students were only provided with an overview of those techniques so they know why/how to use them but not necessarily how to develop them (i.e. they understand the concept of model transformation and know how they could use ATL [16] transformations in their projects and even read them but we don’t ask them to write ATL transformations as part of the course).

The first case study was a reverse engineering scenario where students were asked to use MoDisco [6] to reverse-engineer a Java application and write some (OCL) queries at both the Java model and UML model (generated from the Java one also using MoDisco’s built-in transformations) level to appreciate the power of reverse engineering as a way to understand complex systems. This scenario was important because this is probably one the most common uses of software modeling in practice and, for instance, in the Nantes region, there are important companies that offer this kind of model-based modernization services.

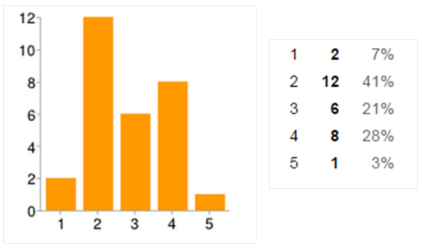
For the larger case study we chose the development of a CRUD-based web application. That is, given a set of requirements regarding the data that a fictitious company needs to manage, students were asked to create all the forms and reports needed to modify/visualize the data through a web application. This scenario was intended to balance the complexity of modeling a web application compared with the amount of code that could be automatically generated from those models. More specifically, students were expected to specify the data model of the application as a UML class diagram, and its navigation model (showing the web pages of the application, the links between them and the actions/events to execute when following a link) as a standard IFML diagram [5]. Given these two models, the WebRatio tool[4] would automatically generate and deploy a fully-functional web application. WebRatio was not the only tool that could be used on this scenario but was chosen because of its level of maturity and due to strong links with the company and researchers behind the tool which would enable us to ask for help – if at all needed. As studied before [2], tool support seems to correlate with more effective MDE teaching.

And, sadly, we did need their help. The reshaping of the course didn't work as we expected and we even had to invite a WebRatio expert to assist the students with the use of the tool (and to be honest, the feeling is that the same would have happened if we had chosen a different tool). At the end of the course, many students were convinced **not** to use MDE again, which was exactly the opposite effect of what we intended. Thus, we did a postmortem analysis to understand what had gone wrong (again), including a very small (but mandatory) survey for the students to answer. In the following we report on the survey questions and their results.



1	**3**	10%
2	**8**	28%
3	**10**	34%
4	**8**	28%
5	**0**	0%

Fig. 1. Answers to Q1 from 5 (very satisfactory) to 1 (totally unacceptable). Total number of answers and corresponding percentages shown on the right.



1	**2**	7%
2	**12**	41%
3	**6**	21%
4	**8**	28%
5	**1**	3%

Fig. 2. Answers to Q2 from 5 (totally sure) to 1 (no way I'm doing that). Total number of answers and corresponding percentages shown on the right.

Q1 – How would you mark the experience of using the code-generation tool?
Figure 1 summarizes the results. Among the most cited (negative) reasons we had: *lots of installation and configuration problems* (students were eventually allowed to work in pairs to make easier for them to have access to at least one machine were the tool was working smoothly), *lack of optimization of the deployed application* (in terms of the size of the generated files specially due to a default set of generated infrastructure code, quite noticeable for a small application like the one in the course), *sudden crashes and corrupted projects*, *good for prototyping but unsure if the method scales*, *lack of*

[4] `http://www.webratio.com/`

documentation and *difficult to customize the code*. There were also some (but fewer) positive comments like *I think this is the tendency of the future* but we always learn more from criticism.

Q2 - If you were working in a software company, how likely is it that you would choose to use some kind of code-generation tool in your next web development project ?

Not very likely according to the results shown in Figure 2. Here a few students mention that *a MDD tool could be used to generate the back-end part and be a great help for database management or a quick and dirty generation for a prototype but for the front-end part, I think it is a waste of time* particularly because they had the feeling they would need to end up modifying lots of the generated code to polish the result since this kind of tool will never be as configurable (at the model-level) as native HTML/CSS/JS code where one can precisely configure every single graphical aspect of the front-end. A couple of students also mentioned that for the kind of scenario they would find the tool useful (this back-end admin-like generator) most programming frameworks nowadays can already generate a simple scaffolding interface from only a database definition which would be good enough and simpler to use. Clearly, this highlights the need to find a sweet spot between language expressiveness and tool simplicity here. Too complex and developers are concerned that they need to invest too much time learning and modeling. Too simple and they will not perceive the benefits.

Q3 – In your opinion, what would make MDD tools more useful and attractive to programmers?

This open question produced good suggestions for some new features for this class of tools, particularly related to having a better and more user-friendly experience with them, like:

1. Being able to build your pages by drag and drop;
2. Ability to change the generated code in a manner that would revert back on the model;
3. Having some common patterns already implemented by default (like login, CRUD tasks, etc. . .);
4. Easier to build multi-language applications;
5. Requests for non-functional aspects like documentation and better compatibility (result of the problems reported in the first question).

The following comment from one of the students is a good summary of a shared feeling among the cohort: *The concept of generating code seems good in itself. However, I had so many problems with the tool I didn't even think I was saving time*. In the next section we discuss some measures that can be put in place to alleviate this situation and ensure that students do not only think that code generation is a good idea but that they are also convinced that it works in practice. These measures will be directed towards addressing the socio-technical factors that caused this perception.

5 Recommendations and Lessons Learned

Based on the previous results, private conversations with colleagues around the world and popular online public discussions[5] we present here a list of lessons learned in the form of recommendations to both MDE instructors (useful for, both, instructors working on companies or in universities and other teaching institutions) and tool vendors to avoid the pitfalls reported earlier.

Recommendations mainly focus on the students perception and experience when interacting with the tools, not on core technical aspects of those tools. For instance, they perceived the tool as generating huge files at the end of the process. This did not really cause any kind of objective efficiency problem when deploying or running the application but they perceived it as a negative value of the tool nonetheless. Therefore, dealing with the reported issues implies considering the sum of the user and the tool as a sociotechnical system that needs to be improved together. Sometimes, we can improve the tool itself but many other times, we need to change the way (the scenario, the conditions,..) students use the tool to give them a more positive experience.

Recommendations for instructors

- Start with a very compelling development scenario. Our CRUD exercise was good to generate a full-fledged working application without requiring complex behavioral modeling but it was too small: the investment required to learn the modeling language and corresponding tooling did not pay off during the project. When exposing first-time learners to MDE, they must feel that they save time thanks to MDE from the very beginning. Therefore we would recommend avoiding toy examples and start with a larger one.
- Change the requirements during the development. Beyond making the exercise more realistic, another way to make MDE more compelling would be to change the requirements of the MDE scenario at the last minute. For instance, in the code-generation case study, we could have asked students to adapt the web application to a number of changes (in the names of the attributes, or their types or even associations between classes) two or three days before the due date. It would be very painful for them to implement those changes directly at the code level (looking for all references to the modified classes anywhere in the code) while doing this at the model level and re-generating should be much easier. To make it even more complex, students could be asked to update not their own projects but the project of another student (again, the goal would be to demonstrate that models can be significantly more understandable and maintainable than code)

[5] It is worth mentioning the discussion in the blog post `http://modeling-languages.com/failed-convince-students-benefits-code-generation/` explaining the first author's preliminary observations that ended up with over four thousand visits and thirty comments (from people with different backgrounds like tool vendors: MetaCase, WebRatio, Softeam, consultants and end-users) and the submission of that same post on reddit `https://www.reddit.com/r/programming/comments/2vehhm/i_failed_to_convince_my_students_about_the/` that brought sixty-five additional comments. Besides these two, additional discussions on these results took place in other forums like a number of LinkedIn groups.

- Use a repetitive scenario. It is known that MDE pays off in the mid-term due to the initial learning curve [1, 10]. Therefore, ideally, students should be asked to complete several similar projects during the course, with the intention of realizing how they their productivity improves at every iteration.
- Compile a set of examples and reference solutions students can refer to and play with. Beyond toy examples, some more complex examples showing that MDE can effectively model and generate non-trivial software should also be provided. In particular, it would be nice to see models simulating popular services like Facebook or Twitter. We believe there should be a community effort to build and share such examples in a public repository. As a side-effect, those same examples could be used as benchmarks to compare the functionality and quality of various MDE tools.
- Keep in mind your target user profile. Introducing MDE to a group of developers in charge of building complex software can be very different than doing so to a group of business users looking for quick – but prototype-level –solutions. WebRatio may work for the former but a solution like Mendix [6] would likely be more appealing to the latter.

Recommendations for tool vendors

- Document, document, document and make sure that documentation is easy to find. The (perceived) lack of proper documentation is what provoked the uprising of students against Acceleo. They did not care whether the tool was the only one properly implementing the OMG standard for model-to-code transformations, they just wanted good documentation to complete the assignment faster.
- Your goal is to hide all underlying technical details. When things go well this is normally the case but, when errors occur in the generation process, tools tend to present users with obscure error messages referencing internal code. This makes finding and correcting the error a daunting task (often involving extensive guesswork as most tools have poor debugging facilities). Even if this happens rarely, when it does, it can cause frustration and disappointment with the tool.
- Offer a well-packaged and standalone installation. For instance, Java-based MDE tools usually ask for the Java environment to use during the installation (even if they have one embedded). This is done to offer more flexibility in the installation process but at the risk of creating unnecessary configuration problems when choosing non-compatible environments. In our case, it turned out that WebRatio was not compatible with the Java version most students had installed as part of another course, so we ended up with over half of the class with an installation of WebRatio they could not use, definitely not a good start.
- Keep up with trends in the software industry. Most software today is an integration of APIs, social components and other kind of services. Students were quickly trying to find components for Google login, Twitter sharing etc. This may not be so relevant for business internal applications but it is for sure something people are now used to find in any web application and will quickly try to replicate when developing their own. Unfortunately, most MDE tools do not offer these features.

[6] `https://www.mendix.com/`

– Favor trust in your tool over everything else. Make sure that your tool is reliable and generates optimal code or at least code good enough that users can trust. If the tool crashes regularly, there is no chance users will believe the code that tool generates is good, they will just assume that a bad tool cannot generate good code. It takes only one bad result to lose their trust.

All these recommendations will not fix by themselves one of the major problems when attempting to convince students about the importance of MDE: the lack of job offers stating MDE as a requirement to apply for the position. Compared to any popular programming language/platform, students feel that MDE is unlikely to boost their employability prospects anywhere nearly as much as becoming experts in, e.g. AngularJS would. This is kind of a chicken-egg problem (increasing the adoption of MDE will create more job opportunities for MDE experts and the other way round) but the previous measures should at least help in moving the market in the right direction.

6 Conclusions

Teaching MDE, and software modeling in general, is considered positive in itself [18] but there is a lot to be done if we wish that teaching to give students a more complete picture of what MDE can do for them in their future professional life. As reported herein, succeeding in this goal requires a careful preparation of a respective course to ensure students have a positive first experience. We have presented a set of lessons learned, which we believe can help other educators in emphasizing the human factors that come into play when preparing such a course.

As further work we plan to replicate these experiments in other educational institutions. In particular, in the near term, we will focus on the e-learning model of the UOC's (`www.uoc.edu`) Faculty of Computer Science, in charge of providing online university education in Information and Communication Technologies since 1997 through its fully virtual campus [20]. Additional validation will be done through collaborations with partners from the network of institutions that teach MDE[7]. A more ambitious plan involves following up on some of the (convinced) students to assess whether the experience made any difference in their approach to software development once they started their professional career. On this, the UOC environment can also help since many of the students are already working professionals seeking additional qualifications.

References

1. R. Acerbis, A. Bongio, M. Brambilla, M. Tisi, S. Ceri, and E. Tosetti. Developing ebusiness solutions with a model driven approach: The case of acer EMEA. In *Web Engineering, 7th Int. Conf. ICWE 2007, Proceedings*, pages 539–544, 2007.
2. S. Akayama, K. Hisazumi, S. Hiya, and A. Fukuda. Using model-driven development tools for object-oriented modeling education. In *Educators' Symposium, 16th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS 2013).*, 2013.

[7] See an (incomplete) list, taken from the set of instructors that declare to be using [3] in courses, here: `https://www.sites.google.com/site/mdsebook/courses`

3. M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.
4. M. Brambilla and P. Fraternali. Large-scale model-driven engineering of web user interaction: The webml and webratio experience. *Sci. Comput. Program.*, 89:71–87, 2014.
5. M. Brambilla and P. Fraternali. *"Interaction Flow Modeling Language"*. The MK/OMG Press. Morgan Kaufmann, 2015.
6. H. Brunelière, J. Cabot, G. Dupé, and F. Madiot. Modisco: A model driven reverse engineering framework. *Information & Software Technology*, 56(8):1012–1032, 2014.
7. D. Budgen, A. J. Burn, O. P. Brereton, B. A. Kitchenham, and R. Pretorius. Empirical evidence about the uml: a systematic literature review. *Software: Practice and Experience*, 41(4):363–392, 2011.
8. M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the 2nd Int. Symposium on Empirical Software Engineering and Measurement*, ESEM '08, pages 2–11, 2008.
9. J. S. Cuadrado, J. L. C. Izquierdo, and J. G. Molina. Applying model-driven engineering in small software enterprises. *Sci. Comput. Program.*, 89:176–198, 2014.
10. O. Diaz and F. M. Villoria. Generating blogs out of product catalogues: An {MDE} approach. *Journal of Systems and Software*, 83(10):1970 – 1982, 2010.
11. B. Dobing and J. Parsons. How UML is used. *Commun. ACM*, 49(5):109–113, 2006.
12. W. J. Dzidek, E. Arisholm, and L. C. Briand. A realistic empirical evaluation of the costs and benefits of UML in software maintenance. *IEEE Trans. Software Eng.*, 34(3):407–432, 2008.
13. A. M. Fernández-Sáez, D. Caivano, M. Genero, and M. R. V. Chaudron. On the use of UML documentation in software maintenance: Results from a survey in industry. In *18th Int. Conf. on Model Driven Engineering Languages and Systems, MoDELS 2015*, pages 292–301, 2015.
14. J. Hutchinson, J. Whittle, and M. Rouncefield. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Sci. Comput. Program.*, 89:144–161, 2014.
15. M. Z. Iqbal, S. Ali, T. Yue, and L. C. Briand. Applying UML/MARTE on industrial projects: challenges, experiences, and guidelines. *Software & System Modeling*, 14(4):1367–85, 2015.
16. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.
17. S. J. Mellor and M. Balcer. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
18. M. Petre. Uml in practice. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 722–731, Piscataway, NJ, USA, 2013. IEEE Press.
19. M. Petre. "no shit" or "oh, shit!": responses to observations on the use of UML in professional practice. *Software and System Modeling*, 13(4):1225–1235, 2014.
20. A. Sangra. A new learning model for the information and knowledge society: The case of the universitat oberta de catalunya (uoc), spain. *The International Review of Research in Open and Distributed Learning*, 2(2), 2002.
21. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.
22. A. Vallecillo. On the industrial adoption of model driven engineering. is your company ready for mde? *Int. Journal of Information Systems and Software Engineering for Big Companies*, 1(1):52 – 68, 2014.
23. J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3):79–85, 2014.