Research Reports in Software Engineering and Management 2015:01 ISSN 1654-4870

Ready for Prime Time, - Yes, Industrial-Grade Modelling Tools can be Used in Education

Grischa Liebel, Rogardt Heldal,
Jan-Philipp Steghöfer
and Michel R.V. Chaudron

CHALMERS | UNIVERSITY OF GOTHENBURG
Department of Computer Science and Engineering

Ready for Prime Time, - Yes, Industrial-Grade Modelling Tools can be Used in Education

Grischa Liebel, Rogardt Heldal, Jan-Philipp Steghöfer and Michel R.V. Chaudron

Report no 2015:01
ISSN: 1651-4870
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden

Telephone + 46 (0)31-772 1000

Göteborg, Sweden 2015

Research Reports in Software Engineering and Management No. 2015:01

Ready for Prime Time, - Yes, Industrial-Grade Modelling Tools can be Used in Education

Grischa Liebel, Rogardt Heldal, Jan-Philipp Steghöfer
and Michel R.V. Chaudron

Department of Computer Science and Engineering
CHALMERS | University of Gothenburg

Gothenburg, Sweden 2015

Abstract

It has been stated that industrial-grade modelling tools are unsuitable for teaching modelling. In this paper, we present our experience with a university course on software modelling. In the first year of the course, we used a commercial modelling tool, in the second year the open-source alternative Papyrus. Both tools are considered to be of industrial grade and used in industry. Our quantitative analysis shows that the industrial-grade modelling tools with all their complexity did not have a negative impact on the students' experience of modelling. This shows that industrial-grade modelling tools can be used in the classroom. We analyse why our experience differs from published accounts and conclude that the availability of a tool champion and tailored instruction material is key. From this, we derive recommendations for teacher support from tool-providers (vendors and open source), research directions for researchers and teachers, and for training efforts in the industry.

1 Introduction

Tool usage and education issues have been identified as major obstacles to the wide-spread adoption of model-driven engineering practices [1]. If taught accordingly at university, a large part of this training effort and novice problems with modelling tools could be reduced. In fact, software modelling is taught in a wide range of university curricula all over the world [2]. In the ACM Curriculum Guidelines for Undergraduate Degree Programs in Computer Science "structural and behavioural models of software designs" is mentioned as one of the core topics in teaching Software Design [3]. Reflecting the current practice in industry [4, 5], UML is commonly used as the modelling language of choice, e.g. in [6, 7, 8].

However, there is a common disagreement with respect to which kind of tooling to use [9]. Industry-grade tools in particular have been described as unwieldy [7] and cumbersome for novice modellers [2], and have consequently been replaced by simplified tools targeted at education, e.g. in [10, 11, 12, 13]. While simple educational tools obviously reduce the learning curve for the students, it takes away the potential benefits they get from being educated in using industrial modelling tools.

In this paper, we present our experiences from teaching a software modelling course to undergraduate students in two different years. We used two different industrial-grade modelling tools: in the first year we used a tool[1] that is offered on commercial basis and used in industry, and in the second year we used the open-source alternative Papyrus[2]. The tools were chosen because they had already been in use at some of our industrial partners and we therefore had a clear use case in industry. In the course of this paper, we will answer the following four research questions.

RQ1: How does support for tool usage affect students' satisfaction?

RQ2: How does the tooling affect the students' view on UML/Modelling?

RQ3: How suitable is Papyrus as a modelling tool in a classroom environment?

RQ4: Is Code Generation an important aspect of teaching modelling?

Over the two years, we collected data from 261 evaluation surveys filled out by the students. The quantitative data is available at `http://grischaliebel.de/data/research/data_primeTime_150715.zip`. The data shows that students clearly see the potential of modelling in general, and of UML in particular. While

[1] Note that the license agreement forbids us to state the name of the commercial tool.

[2] https://www.eclipse.org/papyrus/

the complexity and usability of industrial tools causes some extent of frustration among the students, their evaluation is in general balanced. Additionally, we can see from the free-text comments that issues with tool use can clearly be attributed to a lack of support during the course. However, this can be accommodated in future instances of the course by the teachers, not requiring any changes to the nature of the tooling. Hence, we can conclude that it is possible to use industrial-grade modelling tools in education, without a negative effect on the students' view on modelling. The success depends mostly on the support given within the classroom context - not the support from the tool vendor side.

The remainder of this report is organised as follows. In Chapter 2, we describe related publications both from an educational and an industrial perspective. In Chapter 3, the structure and the contents of both years' courses are discussed in detail. The evaluation data collected from the participating students is presented in Chapter 4, followed by a discussion of our own lessons learned in Chapter 5. From our findings we draw recommendations for future employers, tool vendors and the papyrus community, as well as teachers and researchers in Chapter 6. The paper is concluded in Chapter 7.

2 Modelling Tools in Industry and Education

The suitability and the use of tools is widely discussed in the modelling community, both for industrial use and for education purposes.

Several studies report shortcomings with modelling tools in industrial use, e.g. [14, 15, 16, 17, 5]. Baker et al. state in their case study at Motorola on Model-Driven Engineering that poor tool performance is challenging [14]. Mohagheghi and Dehlen report in their literature review on industrial use of Model-Driven Engineering that the maturity of tool chains is unsatisfactory for a large-scale adoption [15]. In a later case study by Mohagheghi et al., the user-friendliness of tools for Model-Driven Engineering is further mentioned to be challenging [16]. In their industrial survey on Model-Driven Engineering, Liebel et al. name tool integration as one of the key shortcomings of Model-Driven Engineering [5]. Additionally, the survey participants commonly named tool usability as challenging. Whittle et al. report their findings from a case study at multiple companies, specifically targeted at problems with tool usage in Model-Driven Engineering [17]. They conclude that while tools cause a number of problems, organisational issues have to be taken into account as well.

In the educational context, several publications discuss the use of modelling tools or present tooling specifically created for educational purposes. Akayama et al. discuss contextual factors for teaching modelling and choosing tools, such as the amount and the quality of available support for the used tool [9]. However, the authors do not report scientific evidence on which kind of tools should be used. Industry-grade tools in particular have been described as unwieldy [7] and cumbersome for novice modellers [2], and have consequently been replaced by simplified tools targeted at education, e.g. in [10, 11, 12, 13]. Crahen et al. use their simplified UML tool QuickUML for teaching UML in a introductory course on object-oriented programming [10]. Turner et al. argue that available UML tools are targeted at developing code and that tools aimed at education often lack basic features such as undo or redo [11]. They propose their own UML tool, minimUML, which provides only "what is commonly used in beginning programming classes". Ramollari and Dranidis claim that professional UML tools are not suitable for educational purposes and therefore propose their own tool, StudentUML [12]. Soler et al. propose a web-based tool for teaching UML class diagrams, which provides immediate feedback to students and which they use in a database course [18]. All of the authors bring forward arguments for their own tools and against the existing professional and pedagogical tools and propose solutions aimed at their specific course context. However, this also limits their applicability to a broader context. While using industrial modelling tools certainly imposes additional complexity to the course set-up, their large functionality makes them more general and applicable in a wide variety of course contexts.

In summary, there is a large degree of disagreement whether education-specific tools should be used in favour of industrial modelling tools. Additionally, the range of courses in which modelling tools are used is substantial, such as database courses [18], object-oriented programming [10], analysis and design [13], model-driven engineering, requirements engineering, or software architecture. However, the challenges that are brought forward in the educational context, such as limited usability or large complexity, are also commonly reported in an industrial context. Therefore, we argue that the same tools should be used both in education and industry and that both domains should learn from each other.

3 Course Content

The Model-Driven Software Development course lasts eight weeks, with three hours of lectures per week during the first seven weeks. Students are assigned to groups of up to eight people, in which they work on a project that spans all eight weeks. There are three voluntary exams during the course to test the students' knowledge and give them the chance to reflect on their learning. The final grade is mainly based on the project, with the voluntary exams having some influence. The course in the second year was mainly done as in the first year, but we changed the modelling tool from a commercial tool to Papyrus. This was done mainly due to the interest in Papyrus just now, both in the research community and at some of our industrial partners. We heard about recent improvements and increased momentum in Papyrus at Models 2014, and one of the teachers had previously worked with the tool. Therefore, and due to problems with the previously used tool, we decided to introduce the new version of Papyrus into the course. This also changed the way we graded the course: in the first year, due to the commercial tools ability to create executable specifications, grading was based on a demo of the system executing the models students created. In the second year, the demo was based on generated code and the quality of the models played a bigger role.

We used pair teaching in both years [19], which has proven to be extremely suitable for teaching modelling. One of the strengths of pair teaching is that when one teacher is engaged in communication with the students, the other teacher can reflect and give clarifying and complimentary comments. This means that we cover less material in the course, but go deeper into the details. Rather than just going through a lot of modelling syntax during the lecture, we are discussing the strengths and weaknesses of modelling much more as well as the way of thinking behind modelling and design decisions. The main goal of the lectures is to mimic the way we want the students to reason about models and to work together in their groups. This ties into the "design thinking" [20] notion that is promoted by the study programme and is discussed later in this paper.

When developing the course we were influenced by constructive alignment [21]. The students work in our case with more than 80% of the material discussed in the classroom during their project tasks.

When we organised the course we had a few strong constraints on the project: the outcome of all the created models should lead to a working system, the students should be reasonably familiar with the project domain, which should in turn include a mix of structure and behaviour, and students should use an industrial-strength tool.

The reason for the first constraint is that we believe that one cannot teach modelling without producing a working system. We wanted the students to work with the same domain, in order to have a certain amount of control from teaching side, but have a lot of freedom within that domain. We did consider using a domain from our industrial partners, but found that to be a less than ideal solution since it would require spending too much time on learning the domain, thus taking the focus away from learning modelling. We decided that a hotel booking system would be good compromise, since it is a real domain that most students are somewhat familiar with. All groups were required to handle at least the booking, check-in and check-out use cases. Additionally, the system was required to be responsible for finding free rooms given a set of customer requirements such as start date, end date and type of rooms. Otherwise, the students were free in extending the system in any way they wanted. The students were expected to do anything from requirement elicitation, creating analysis and design models, to a running system in the end.

All groups received 45 minutes of supervision a week in the first year and 35 minutes in the second year. We had three supervisors in the first year and four supervisors in the second year, all experts in modelling. We tried to align our feedback by having a one-hour meeting among the supervisors every week. The supervision focused mainly on modelling, and not the tool. In both years, only one supervisor was giving tool support, decoupled from the supervision sessions. We thought it was crucial to have an expert in the used modelling tool, and we would not have done a course without that. The tool expert helped all students with questions within a working day, and in the second year also demonstrated the tool live in the classroom. However, due to the allocated teaching budget, we had only a very limited number of hours for tool support, amounting to an average of 11 minutes per student in the second year. Also, the live demo was less than 2 hours. We do not have fixed numbers for the first year, but they should lie in the same range. Hence, the students did not get substantial help in terms of time, but very high quality help just when they needed it and aimed directly at the course content. General tool problems, which many students reported, were discussed in the form of screen casts or short tutorials on the course home page.

4 Evaluation

In both years, we distributed an evaluation survey among the students during the final project presentations. The survey covers the different aspects of the course, such as the lecture content, lecture style, projects, and tooling aspects. The students filled out the survey before receiving their grade, with a number of questions regarding the grading being left until after the grading. In year 1, we had 135 registered students and 164 in year 2. Out of these, 110 surveys were returned in the first year and 151 in the second, corresponding to return rates of 81.5% and 92.1%. It is important to note that filling out the survey was entirely voluntary and anonymous. In the following, data from year 1 is depicted in the red bar charts and data from year 2 is depicted in the blue bar charts.

As discussed in Chapter 3, the course contents in both years were nearly the same. The main change between the years was the used modelling tool and the support which we gave for it. Hence, the difference in student satisfaction between the two years gives interesting insights into **RQ1**, "How does support for tool usage affect students' satisfaction?". In the first year, we were not able to prepare enough support material or familiarise ourselves with the tool due to installation and licensing problems. In the second year, the support for Papyrus was much stronger and specially tailored towards the course. Already during the second course, we could observe that this improved the students' satisfaction compared to the previous year tremendously. This observation is also reflected in our data. The students' views differ with respect to whether the tools can be useful for modelling large and complex systems. On the one hand, in year 1 (red bars in Figure 4.1), the distribution is balanced with both agreement and disagreement. In year 2 (blue bars in Figure 4.1) on the other hand, the answers tend to be more positive. However, the difference between the two groups is not statistically significant. One could argue that the usefulness of the used modelling tool alone is not a good indicator for the students' satisfaction, especially towards the intended goals of the course. Therefore, we also asked the students about the usefulness of UML in particular. The results for both course years are depicted in Figure 4.2. Similar to the previous question, the results are generally very positive, with the students in the second year being significantly more positive than the students in year one (two-sided Fisher Test, $p < 0.01$). This result shows that the change of tooling impacted the students significantly. As the students stated that adding functionality to the models was similarly easy in both tools once they became familiar with them, as depicted in Figure 4.3, we believe that this change can be attributed to the given support. The general satisfaction with the course also increased significantly from the

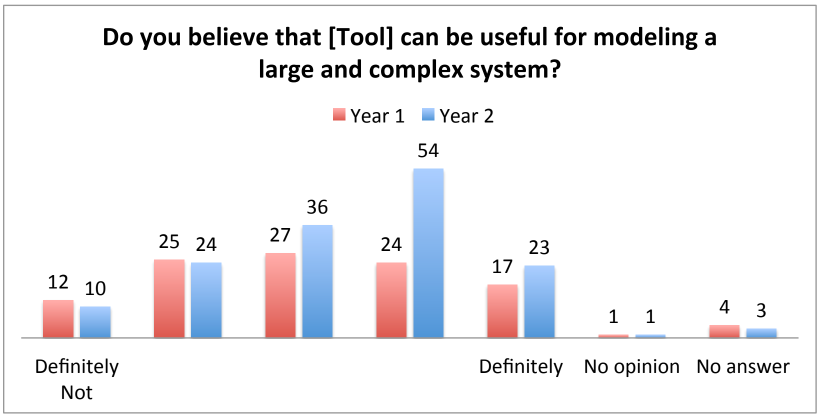

Figure 4.1: Usefulness of Tool

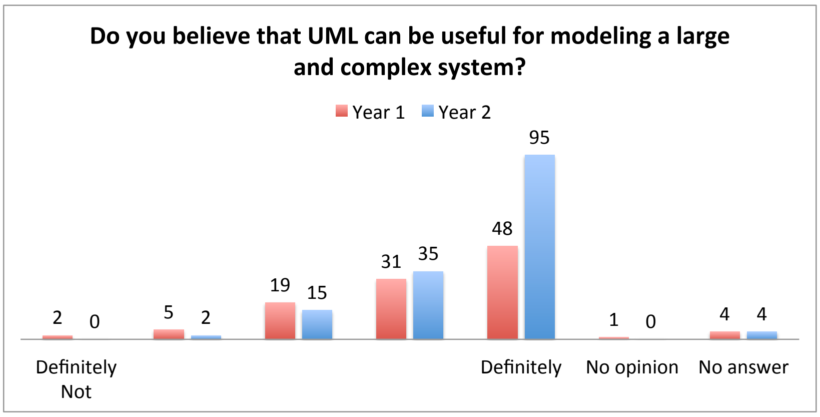

Figure 4.2: Usefulness of UML

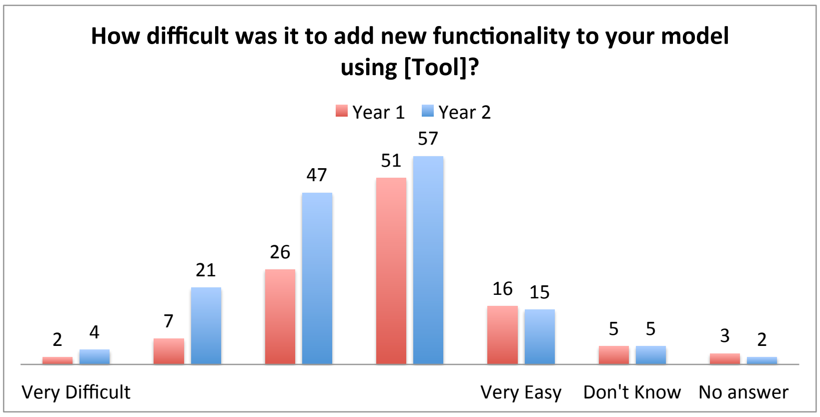

Figure 4.3: Easiness of Adding Functionality

first year to the second (two-sided Fisher Test, $p < 0.01$). However, while this could be attributed to the given tool support, there are too many factors which could possibly affect this figure in order to purely attribute it to the support, e.g. the change in teachers or the workload and quality of other courses running in parallel.

Research Question **RQ2**, "How does the tooling affect the students' view on UML/Modelling?", is directly related to **RQ1**. As stated above, the change of tooling from one year to the other caused a significant increase in how positive students viewed UML modelling. This, we attribute mostly to the increased support in the second year, not to the tooling alone. This means that given the right support, most tools are realistic alternatives. Figure 4.4 shows this change from one tool to the other. While we can observe that the subjects in the second year were more positive towards modelling, it is also important to note that the question was asked in a slightly different way. However, both figures indicate that modelling is indeed seen positive by the students, regardless of the tool we used.

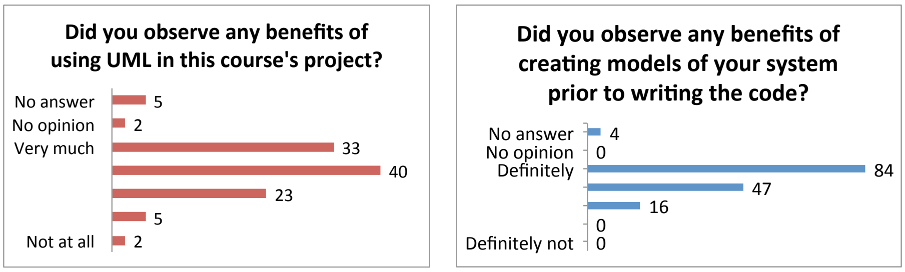

Figure 4.4: Benefits of Models

While not entirely identical, we asked similar questions regarding the impact of the modelling tool in both course years. The results for these questions are depicted in Figure 4.5. Here, it is notable that in year 2, few people answered the extremes, i.e. the lowest and highest values on the five-point Likert scale. We see the fact that students in both years are relatively neutral as a positive indicator, as this means that industrial-grade tools are a realistic alternative to specialised education tools. At the same time, we have to acknowledge their complexity and can therefore not expect an overly positive evaluation.

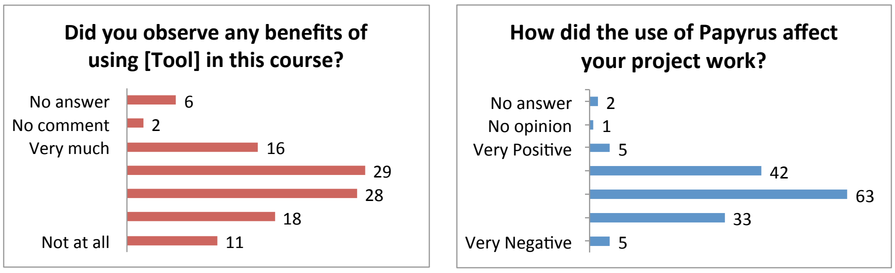

Figure 4.5: Tool Impact

Research Question **RQ3**, "How suitable is Papyrus as a modelling tool in a classroom environment?", targets only Papyrus and therefore the second course year. Given the attention Papyrus currently receives in the modelling community, we were curious how it performs in a classroom environment. As shown in Figures 4.1, 4.2, 4.3, 4.4, and 4.5, Papyrus is at least comparable to the commercial tool we used previously in the course. Additionally, we asked about the difficulty to learn Papyrus (Figure 4.6). Again, the answers are rather balanced with students both stating that it was difficult and that it was easy to learn Papyrus. Altogether, this is good news for modelling education, as it means that we can use open source software in modelling education and achieve similar results as with commercial industrial-grade tools. Additionally, we asked whether the benefits of creating the models outweigh the effort needed to create them. This is particularly relevant as the effort of creating models is often stated to be too high (cf. [5, 4]). Also, a tool can be very efficient and usable, but still the effort to use it might not be justified. The answers are summarised in Figure 4.7. Only very few students answer that the effort is too high compared with the outcomes and most students state that the advantages possibly or definitely outweigh the effort to create them.

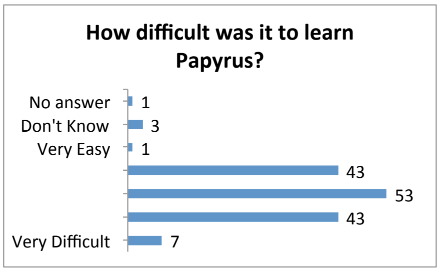

Figure 4.6: Learning Papyrus

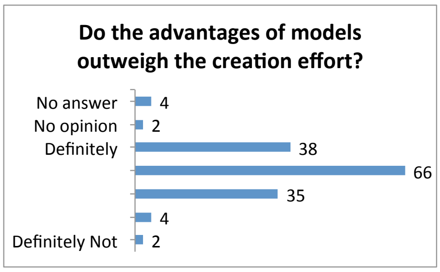

Figure 4.7: Modelling Effort

As discussed in Chapter 3, we used executable models in the first year. However, we often felt like the students did not really understand their models and had difficulties relating them to anything they had seen or learned before. Therefore, in the second year we introduced code generation, with the hope that students would make the connection between their class diagrams and the resulting code. Research Question **RQ4**, “Is Code Generation an important aspect of teaching modelling?”, is aimed at evaluating the outcome of this introduction. We used the EMF[1] code generation capabilities, as they support re-generation and protected areas. However, the used code generation mechanisms are mainly aimed at generating model editors and therefore generate code which might be difficult to use and to understand at first. Therefore, we asked a number of questions evaluating the use of code generation during the course. Overall, there was no agreement on how difficult it was to generate code ($\mu = 3.14, median = 3, \sigma^2 = 1.08$, on a 5-point Likert scale from “1 - Very Difficult” to “5 - Very Easy”), on how difficult it was to use the generated code ($\mu = 3.05, median = 3, \sigma^2 = 1.06$), or on how the use of code generation generally affected the project ($\mu = 3.07, median = 3, \sigma^2 = 1.0$, on a 5-point Likert scale from “1 - Very Negative” to “5 - Very Positive”). From the free-text comments, we can tell that there was a lot of dissatisfaction regarding the use of code generation. This was mainly attributed to the amount of generated code.

[1] http://www.eclipse.org/modeling/emf/

Students stated that the generated code was “heavy”, “smelly”, or “complex”. Additionally, we can tell from discussions with students, that some students felt like they were restricted in their freedom to program in a certain way, as they had to follow the code patterns prescribed by EMF. On the positive side, students mentioned that the models were always consistent with the code, that re-generation of code was possible and easy to do, and in general the fact that you could generate code from your model. Summing up all four research questions, students clearly see the benefit of UML and modelling in general, regardless of the used tool. Additionally, the evaluation shows a balanced view on the tools themselves. This view confirms our impression that using industrial-grade tools, such as Papyrus, and approaches not specifically tailored towards the course, such as EMF code generation, are in fact suitable for use within a classroom environment.

5 Lessons Learned

While the previous chapter detailed the results of our quantitative analysis, the following paragraph outline our subjective experience and the lessons we have learned in the discussions between the teachers and the reflections of each teacher individually. Many of them are related to our research questions but others are more specific to the setup of the course and the learning outcomes we wanted to achieve. When appropriate, we base our observations on the free text the students provided in their evaluations.

Teaching modelling vs. teaching design This course is part of a programme that aims at educating students to become software engineers. As in most engineering programmes, "design thinking" [20] is one of the key learning outcomes overarching all terms of the programme. One of the issues that has been discussed repeatedly between the teachers of the course is the separation between modelling and design. When assessing the students work, it is impossible to differentiate between these aspects. A "good" model is always one that depicts a good design, avoids waste, is flexible and comprehensible. It must be noted, though, that the quality of the design is completely independent of the tool used to create the model. As long as the tool does not prevent the efficient creation of the model, it should have no impact on the quality.

In fact, our ongoing discussions regarding the student assessment showed that these aspects are inextricably linked with each other. The same aspects were evident in the personal weekly interaction with the students. Much of the discussion in the sessions was focused on questions of design. Rather than discuss the syntax and semantics of the modelling language, students were interested in discussing their design choices and the impact they have on a future implementation. The modelling itself seemed to be intuitive enough. This is a strong argument for the fact that the tool support is good enough to move the focus away from practical issues (**RQ1** and **RQ2**).

Good tool support and good teaching is not enough One of the main differences between the two course years is the amount and the quality of tool support we could give the students. In the first year, our own support was limited, as we encountered late licensing and installation problems. This severely reduced the teachers' time to familiarise themselves with the tool and the changes since previous versions. Additionally, we could not prepare enough support material and had to rely on online or old material. In the second year, one of the teachers had a deep knowledge of Papyrus, at least within the limits of the covered topics, and enough time prior to course start to prepare lecture material specifically covering tool use. Both the lectures and additional tutorials, when needed, were consequently recorded as screen casts and made available to the students. From our discussions with the students throughout the courses and from their free-text answers in the evaluation surveys, we can clearly see the difference between the two years. In the first year, we received lots of complaints regarding the tooling in general. Students stated that it is *"very hard to learn in the beginning"* or that it causes *"initial confusion"*. Especially prominent were voices that requested more documentation or complained about the lack or the quality of documentation. It must be noted, however, that this general dissatisfaction with the tool was not as evident in the quantitative data as discussed in Chapter 4. This has relevance for **RQ1**, **RQ2**, and **RQ3**. In the second year, the comments regarding Papyrus were much more targeted at the surroundings, especially things we had not covered in the form of screen casts or in lectures. These were, for example, the code generation or the code resulting from it, or the use of version control with models. Additionally, we received positive comments regarding the provided tutorials which were not just general tutorials, but specifically aimed at the course and the covered material (**RQ1**).

From these observations, we learn that it is essential to have teachers or staff, who both know the taught course and the tooling well enough to give dedicated support. It is not enough to have good tool support and a good course, if the tool supported is too generic and not specific to the course contents. In fact, it might be a misconception that tools specifically developed for teaching purposes are better-suited for teaching due to the reduced complexity. The actual reason could be that the developers of these tools are often identical with the teachers giving the courses from which these claims stem. Hence, they know their tool very well and, additionally, how it should be used in the right way within their course.

Losing control of your code In the second year, several groups of students complained that the generated code was too large, followed a bad design, or was cumbersome to work with. The code generator creates an interface and a concrete class for each class in the design class diagram. While this practice corresponds to modern enterprise software development standards, the students were confused and doubted the purpose of this. More guidance before the stage where code is generated would therefore be helpful. Interestingly, these were often groups of very good students, who did not have the problem of understanding the topic in itself. These students did not see the benefit of generating code, but were frustrated that they could not write it in the way they wanted to. Similarly, in both course years, we often noticed that especially students who really enjoy programming have a hard time to see the benefits of modelling in general. Of course, the quality of the generated code directly reflects the quality of the design. Therefore, for some teams, the design of the system got significantly better once the code generation was used. Apparently, those students that struggled with a good design but are able to code well realised their mistakes as soon as they saw the code that was generated. This caused them to go back and fix the issues with the models, regenerate the code, and so forth. It might therefore be a good idea to let the students generate code sooner in the progress of the course to realise this advantage earlier, allowing for an iterative approach as discussed, e.g., in [20]. We thus also believe that **RQ4** can be answered in the positive.

Over-customisation is dangerous In the second year, we used the customisation capabilities of Papyrus in order to hide a number of menu elements for diagram types and palette items which we deemed unnecessary for the course. With this, we wanted to facilitate the use of Papyrus for novice modellers, similarly to the initiative *Papyrus for Education*[1]. However, this lead to a number of challenges throughout the course. Firstly, a few items in the palette were simply forgotten, leading to strong restrictions with some diagram types that were only discovered by students. While this was obviously a mistake from the customiser's side, it is almost impossible to ensure that you have not forgotten any items. Additionally, the short time period in which the course runs makes it infeasible to accommodate for any pilot or trial periods. Another problem with respect to tool customisation was caused by the number of involved teachers. As we did not always achieve a perfect synchronisation, it happened that teachers told students to use certain diagram elements, which we did not anticipate when we customised the tool. Therefore, these elements were not present in the customised version, which lead to confusion and, sometimes, frustration among the students. This frustration was solved, however, when teachers told students to switch back to the full version of Papyrus, showing again that the tool in its current form is usable in the classroom (**RQ3**). From this experience, we derive that you should only customise your tools if you are in control of all the course moments and all involved teachers or teaching assistants. Additionally, if we are already unable to provide a customised version fitting a single course over which we have a large degree of control, this means that a simplified version of Papyrus, or any other tool for that matter, is unrealistic. Every teacher has her own style of teaching modelling and will focus on different aspects.

[1] https://wiki.eclipse.org/Papyrus_for_Education

6 Recommendations

Our experience and our analysis has revealed a number of interesting aspects discussed in the previous chapters. We also feel that they put us in a position to give recommendations for the vendors of commercial modelling tools that want to see their tools used in academia, for the Papyrus community, for teachers and researchers that are working with modelling tools, and for the future employers of our students who at the same time are the industrial users of the tools we discuss. These recommendations are drawn from our subjective experience, but based on the available data.

For Tool Vendors

In general, tool vendors are interested to have students use their products in an educational setting in the hope that they will carry the knowledge into their future jobs and have their future employers adopt the now-familiar tool. It turns out, however, that the experiences the students have with the tool are critically linked to how the tool was presented and thus how it was perceived. This perception is, in turn, directly linked to the competence of the teacher in working with the tool.

The teachers are however usually left with very little support from the vendor's side. In the case of the course discussed in this paper, the teachers were so involved with issues regarding licensing and installation of the commercial tool that it was impossible for them to establish the level of knowledge necessary to provide the students with good support. These issues are also immensely time consuming so that the time budget the teacher had to work with the tool and the students was severely depleted.

In essence, if the teacher is not competent, she will be of no help to the students, and this will leave the students frustrated over the tool. Modelling tools are necessarily complex and difficult to pick up. If a company thus puts additional burdens, e.g., due to the way they handle licensing, in the way of the teacher, the desired effect will most likely not be achieved. Instead, the tool vendor should have a vested interested in making sure that the teacher is competent enough and has sufficient time to become the champion of the tool and hereby create a good experience for the students who will then take this experience with them into the industry. This can, e.g., be achieved by providing specialised resources as discussed below.

For the Papyrus Community

A similar line of thinking can be applied for the Papyrus community. We have demonstrated that the tool is viable in an educational setting if the general set-up is right. However, as discussed in the lessons learned, we believe that providing a simplified version for education is the wrong approach. Instead, we believe there are three more promising avenues to make Papyrus more accessible:

1. Provide example-driven, up-to-date documentation of the tool. One of the hypotheses we had going forward with Papyrus was that the user community and the available online documentation would be helpful resources for the students. It turns out that this was not the case. The partially out-dated information available online proved to be difficult to work with since students do not have the ability to distinguish which information is current and which is not, causing frustration for the students. As a response to this, the students did however not participate in the Papyrus forum, e.g., but addressed their questions to the teachers. This can surely be attributed to the availability of the tool champion but can also stem from a reluctance on the students' side to interact with a community they know very little about.

2. Provide specialised resources for teachers. The Papyrus community would benefit from more competent teachers the same way the tool vendors would benefit. Therefore, targeted resources for teachers that can be used to achieve a high level of competency and that can be used to introduce the tool in the classroom would be helpful. Concretely, prepared scenarios that can be used to introduce the tool with a realistic setting in live demos would be helpful. Screen casts that show the relevant features of Papyrus for different levels of expertise could supplement the material prepared by teachers. Finally, the training of teachers to become Papyrus "champions" in the classroom might even be part of a business model should the Papyrus community choose to complement their open source offerings with a commercial service.

3. Simplify the process of customising Papyrus. There are facilities right now to create a customised version of Papyrus that limit the modelling elements that can be used. As discussed before, we tried to make use of these facilities with mixed results. However, if it was possible to tailor Papyrus more easily, it would become easier to adapt the tool to each specific course. A teacher could then create, prototype, test, and refine a specialised version of Papyrus that fits the requirements of the course at hand without having to mess with XML-files and such. Instead of trying to provide a "one size fits all" solution, this would allow targeted adaptation aligned with the intended learning objectives, the concrete projects, and the progress of the lectures.

We do not believe that Papyrus has any properties that make it stand out as an ideal teaching tool just yet. In fact, our evaluation showed that once students were confident with the tool, they actually found it easier to add new functionality with the commercial tool used in year 1 (cf. Fig. 4.3). We do believe, however, that Papyrus has the potential to become a leading educational tool.

For Teachers and Researchers

Our findings indicate that industrial-grade modelling tools can be used in the classroom successfully and that the arguments brought forward to, e.g., introduce a new tool specialised on education (cf., e.g., [11, 12]) are not valid in all cases. Instead of spending resources on the development of yet another modelling tool, we instead suggest to invest these resources in becoming a champion for an existing tool and providing the level of support to the students they require. For example, the Papyrus tool is primarily targeted at being used as a tool in software development. Teachers could invest in developing educational materials such as 'wizzards' that offer help to novice designers ([22]) or 'intelligent tutoring systems' that integrate seamlessly with Papyrus. Additionally, free-text comments from the second year show us that students appreciate that they are being taught a tool which is used in industry and might prove useful for them later on. As a teacher, on the other hand, the Papyrus community and especially the forums offer a great way to quickly solve problems students encounter and engage the developers directly.

Finally, we strongly believe that tooling in modelling education should not be seen as totally separated from tooling in industry. If approached in the right way, education could learn from industry and vice versa. We therefore encourage further research on how to connect modelling education and modelling in industry, not exclusively with a focus on tooling.

For Future Employers/Industry

Even if a modelling tool is introduced as part of a course in the way we did it, students can not reasonably be expected to retain the tool knowledge for extended periods of time. Instead, the students learn a more abstract set of skills whose application the tool facilitates. Since the modelling course is part of a programme in which "design thinking" [20] is central, future employers should thus expect students to have the ability to think in the abstract terms required in the design domain and at least be able to quickly pick up similar modelling tools.

While our findings stem from the classroom, there is literature that supports a generalisation into the industry. "Change champions" [23] are known to facilitate the adoption of new technologies in much the same way our tool champion did. The key issue is the kind of "just-in-time" support that the champion can offer , which gives users of a tool a very direct feedback line that is not available if the tool is introduced as part of a training workshop as they are often used in industrial settings. In addition, the support given by the champion is always related to the domain and the concrete problem the users are working in, rather than the abstract support a tool vendor can give. We therefore hypothesise that the introduction of tools in industry can benefit from focused support through a champion just as much as the students did in our classroom setting.

7 Conclusion and Future Work

In this paper, we have demonstrated that industrial-strength modelling tools can be used in a classroom setting. However, a ‘tool champion’ is required for giving just-in-time support. Our analysis shows that the tool does not negatively impact student satisfaction and that the students are able to see the benefits of modelling even though they are using a complex tool.

Our analysis of student responses and our own experience also showed that there is a need for improvements not in the actual core tools (’modelling’), but the supporting tools. This pertains, e.g., to version control of models, code-generation, and customisation capabilities. With regard to our research questions, we show that support for tool usage greatly affects student satisfaction (**RQ1**), that a good experience with the tool has a positive influence on the students’ view of UML and modelling in general (**RQ2**), that Papyrus is suitable for a classroom environment given the right level of support (**RQ3**), and that code generation is essential to understand modelling (**RQ4**).

Our recommendations pick up on the quantitative data and the lessons learned. We suggest that tool vendors and the Papyrus community focus on providing specialised resources to teachers that help them become tool champions. We also recommend that researchers and teachers invest the time to achieve this status before dismissing such tools as unusable. Finally, we suggest that industry also benefits from a tool champion and that a professional tool smith supporting a university course can make costly in-company training obsolete.

In future work, we would like to conduct a study on knowledge retention to see how well tool knowledge is retained after training. We suspect that the specific knowledge of how the tool is used is retained only for a short period of time, but that the more abstract knowledge of how to create models is retained much longer. This line of research is complementary to one in which we would like to investigate if the complexity of industrial-strength modelling tools negatively affects knowledge retention and the ability to create good models.

Bibliography

[1] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, J. Hill, J. Kienzle, M. SchÃ¶ttle, F. Steimann, D. Stikkolorum, and J. Whittle, “The relevance of model-driven engineering thirty years from now,” in *Proc. of ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems.* Springer International Publishing, 2014, vol. 8767, pp. 183–200.

[2] R. F. Paige, F. A. Polack, D. S. Kolovos, L. M. Rose, N. Matragkas, and J. R. Williams, “Bad modelling teaching practices,” in *ACM/IEEE 17th Int. Conf. on Model Driven Engineering Languages and Systems – Educators Symposium*, 2014.

[3] ACM, “Curriculum Guidelines for Undergraduate Degree Programs in Computer Science,” https://www.acm.org/education/CS2013-final-report.pdf, Apr. 2015.

[4] M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio, “Preliminary Findings from a Survey on the MD* State of the Practice,” in *Proc. of 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Sept 2011, pp. 372–375.

[5] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, “Assessing the state-of-practice of model-based engineering in the embedded systems domain,” in *Proc. of ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems.* Springer International Publishing, 2014, pp. 166–182.

[6] E. Astesiano, M. Cerioli, G. Reggio, F. Ricca, and C. Unita, “A phased highly interactive approach to teaching uml-based software development,” in *ACM/IEEE 10th Int. Conf. on Model Driven Engineering Languages and Systems – Educators Symposium*, vol. 7, 2007, pp. 9–18.

[7] T. Lethbridge, G. Mussbacher, A. Forward, and O. Badreddin, “Teaching UML using umple: Applying model-oriented programming in the classroom,” in *Proc. of 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE T)*, May 2011, pp. 421–428.

[8] G. Engels, J. Hausmann, M. Lohmann, and S. Sauer, “Teaching UML Is Teaching Software Engineering Is Teaching Abstraction,” in *Satellite Events at the MoDELS 2005 Conference.* Springer Berlin Heidelberg, 2006, vol. 3844, pp. 306–319.

[9] S. Akayama, B. Demuth, T. C. Lethbridge, M. Scholz, P. Stevens, and D. R. Stikkolorum, "Tool use in software modelling education," in *ACM/IEEE 16th Int. Conf. on Model Driven Engineering Languages and Systems – Educators Symposium*, vol. 1134. CEUR-WS.org, 2013.

[10] E. Crahen, C. Alphonce, and P. Ventura, "Quickuml: A beginner's uml tool," in *Companion of 17th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications.* ACM, 2002, pp. 62–63.

[11] S. A. Turner, M. A. Pérez-Quiñones, and S. H. Edwards, "minimuml: A minimalist approach to uml diagramming for early computer science education," *J. Educ. Resour. Comput.*, vol. 5, no. 4, Dec. 2005.

[12] E. Ramollari and D. Dranidis, "Studentuml: An educational tool supporting object-oriented analysis and design," *Proc. of 11th Panhellenic Conference on Informatics*, pp. 363–373, 2007.

[13] D. R. Stikkolorum, C. Stevenson, and M. R. V. Chaudron, "Assessing software design skills and their relation with reasoning skills," in *ACM/IEEE 16th Int. Conf. on Model Driven Engineering Languages and Systems – Educators Symposium*, vol. 1134. CEUR-WS.org, 2013.

[14] P. Baker, S. Loh, and F. Weil, "Model-Driven Engineering in a Large Industrial Context – Motorola Case Study," in *Proc. of ACM/IEEE 8th International Conference On Model Driven Engineering Languages And Systems.* Springer Berlin Heidelberg, 2005, vol. 3713, pp. 476–491.

[15] P. Mohagheghi and V. Dehlen, "Where Is the Proof? – A Review of Experiences from Applying MDE in Industry," in *Proc. of 4th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA).* Springer Berlin Heidelberg, 2008, vol. 5095, pp. 432–443.

[16] P. Mohagheghi, W. Gilani, A. Stefanescu, M. Fernandez, B. Nordmoen, and M. Fritzsche, "Where does model-driven engineering help? Experiences from three industrial cases," *Software & Systems Modeling*, vol. 12, no. 3, pp. 619–639, 2013.

[17] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial adoption of model-driven engineering: Are the tools really the problem?" in *Proc. of ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems.* Springer Berlin Heidelberg, 2013, vol. 8107, pp. 1–17.

[18] J. Soler, I. Boada, F. Prados, J. Poch, and R. Fabregat, "A web-based e-learning tool for uml class diagrams," in *Proc. of IEEE EDUCON - Education Engineering Conference.* IEEE, 2010, pp. 973–979.

[19] H. Burden, R. Heldal, and T. Adawi, "Pair Lecturing to Model Modelling and Encourage Active Learning," in *Proceedings of 11th Active Learning in Engineering Workshop (ALE)*, Copenhagen, Denmark, June 2012.

[20] C. L. Dym, A. M. Agogino, O. Eris, D. D. Frey, and L. J. Leifer, "Engineering design thinking, teaching, and learning," *Journal of Engineering Education*, vol. 94, no. 1, pp. 103–120, 2005.

[21] J. Biggs, "Enhancing teaching through constructive alignment," *Higher education*, vol. 32, no. 3, pp. 347–364, 1996.

[22] H. Wood and D. Wood, "Help seeking, learning and contingent tutoring," *Computers & Education*, vol. 33, no. 2â3, pp. 153–169, 1999.

[23] J. M. Howell and C. A. Higgins, "Champions of change: Identifying, understanding, and supporting champions of technological innovations," *Organizational Dynamics*, vol. 19, no. 1, pp. 40–55, 1990.