Experiences of Teaching Model-based Development

Kevin Lano[1], Sobhan Yassipour-Tehrani[1], Hessa Alfraihi[1]

[1]Dept of Informatics, King's College London, Strand, London, UK

Abstract. Since 2012 we have been teaching agile development and model-based development (MBD) in undergraduate courses at King's College London. In this paper we analyse the results of practical courseworks in agile MBD, taken from 2013-14 and 2014-15. We identify the factors which have contributed to the success or failure of students to apply theoretical knowledge to these practical problems, and we consider how educational techniques can be improved in this area.

Keywords — Model-based development (MBD) education; Model-based development; Agile development education.

1 Educational context

The case studies described here are the practical coursework component (counting for 15% of the overall marks) in a second year compulsory undergraduate course "Object-oriented Specification and Design" (OSD) in Computer Science. This course is our students first exposure to UML and to concepts of software specification and design: the first year of the degree programme is dedicated to mathematical foundations and introductory programming in Java. The OSD course covers: UML modelling (class diagrams, state machines, OCL); concepts of specification and design; design patterns; model transformations and MBD; agile development. The coursework takes place in the last four weeks of term, after all necessary theoretical material has been taught. As well as an exercise in MBD, the coursework is an exercise in effective teamwork and project management. Students already have experience of team working on the concurrent Software Engineering Group project (SEG). For OSD, the students are assigned randomly to teams of between 8 to 10 people. The teams are instructed to specify and implement the coursework project using the UML-RSDS agile MBD toolset [3]. They also need to produce a project report describing their work. Teams are assessed on the quality of the implementation, of the report, and on their team management and organisation. Teams are advised to select a leader, and to apply an agile development process, although a specific process is not mandated. A short requirements document is provided, and links to the UML-RSDS tools and manual. Each week during the coursework there is a one hour timetabled lab session where teams can meet and ask for help from postgraduate students who have some (but not expert-level) UML-RSDS knowledge.

In Section 2 we give an overview of UML-RSDS. Sections 3 and 4 describe the courseworks and their outcomes, and Section 5 draws conclusions about the results of these projects. Section 6 discusses related work, and Section 7 gives conclusions.

2 UML-RSDS

UML-RSDS uses UML class diagrams and use cases to specify the data and behaviour of systems at a high level of abstraction. From these specifications, designs and executable code (in Java) can be automatically generated. Developers using UML-RSDS should focus on defining and improving the specification, and should never need to manually modify the generated code.

For example, one solution to the 2013-14 coursework below could consist of the class diagram and use cases of Figure 1.

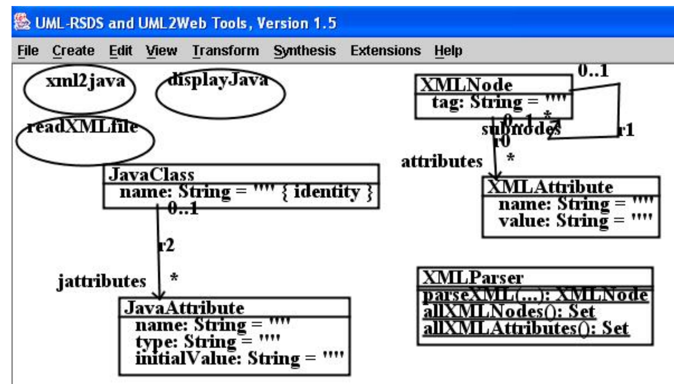

Fig. 1. FIXML system specification in UML-RSDS

The class *XMLParser* is an external class (its code is provided by handwritten Java). The *xml2java* use case has the initial postcondition constraint:

```
XMLNode::
  JavaClass->exists( jc | jc.name = tag )
```

The operational interpretation of this postcondition is that it creates a Java class for each XMLNode (task 2a of the coursework). It can be read logically as "For all XMLNode instances, there is a JavaClass with the same name". For task 2b there is the postcondition:

```
XMLNode::
  att : attributes & jc = JavaClass[tag] &
  att.name /: jc.jattributes@pre.name@pre  =>
```

```
JavaAttribute->exists( ja | ja.name = att.name &
        ja.type = "String" & ja.initialValue = att.value &
        ja : jc.jattributes )
```

This maps all XML attributes of a given XML node *self* to program attributes of the program class $JavaClass[tag]$ corresponding to *self*. For each pair of an XML node *self* and attribute $att : attributes$ a new JavaAttribute is created and added to $JavaClass[tag]$.

Both metamodels and application models are represented using UML-RSDS class diagram notation. Instance models are represented as text files with a simple format to identify (i) that an object belongs to a type $x : T$; (ii) that a single-valued object feature has a particular value $x.f = v$; or (iii) that an object belongs to a collection-valued feature $x : y.f$. The code generated from a UML-RSDS system specification can read and write instance models in this format, alternatively, as in Section 3, instance models can be read or written in XML formats.

3 Coursework 2013-14: FIXML code generation

This problem was based on the project described in [4]. Financial transactions can be electronically expressed using formats such as the FIX (Financial Information eXchange) format. New variants/extensions of such message formats can be introduced, which leads to problems in the maintenance of end-user software: the user software, written in various programming languages, which generates and processes financial transaction messages will need to be updated to the latest version of the format each time it changes. For this coursework we restricted attention to generating Java, C# and C++ class declarations from messages in FIXML 4.4 format, as defined at http://fixwiki.org/fixwiki/FPL:FIXML_Syntax, and http://www.fixtradingcommunity.org.

The solution transformation should take as input a text file of a message in XML FIXML 4.4 Schema format, and produce as output corresponding Java, C# and C++ text files representing this data.

The problem is divided into the following use cases:

1. Map data represented in an XML text file to an instance model of the XML metamodel.
2. Map a model of the XML metamodel to a model of a suitable metamodel for the programming language/languages under consideration. This has subtasks: 2a. Map XML nodes to classes; 2b. Map XML attributes to attributes; 2c. Map subnodes to object instances.
3. Generate program text from the program model.

An XML parser written in Java was provided to assist students with task 1.

Approximately 120 students were on the OSD course in 2013-14, and these were divided into 12 teams of 10 students each.

The coursework involves research into FIXML, XML, the UML-RSDS tools and into C# and C++, and the definition of use cases in UML-RSDS using

OCL. Students need to understand the concept of metamodelling of XML and of programming languages. None of these topics had been taught to the students. Scrum, XP, and an outline agile development approach using UML-RSDS had been taught, and the teams were recommended to appoint a team leader. A short (5 page) requirements document was provided.

3.1 Outcomes

The outcome of the coursework is summarised in Table 1.

Teams	*Mark range*	*Result*
5, 8, 9, 10	80+	Comprehensive solution and testing, well-organised team
12	80+	Good solution, but used manual coding, not UML-RSDS
4, 7, 11	70-80	Some errors/incompleteness
2, 3, 6	50-60	Failed to complete some tasks
1	Below 40	Failed all tasks, group split into two.

Table 1. Coursework 1 results

The main technical problems encountered by teams were:

- Inability to understand the requirements of the problem, or to understand how to use UML-RSDS to solve it. Seven of the 12 teams achieved a good or adequate understanding of UML-RSDS, however this took considerable time and effort. The other five teams failed to achieve an adequate level of understanding.
- Difficulty in learning new programming languages (C++ and C#) sufficiently to be able to generate correct code in them.

A small minority (perhaps 5%) of the students were enthusiastic about the code-generation concept, once they understood how to use UML-RSDS correctly. A larger set of students (perhaps 40%) were able to understand the concepts and technologies sufficiently to solve the problem, whilst the remainder either did not understand the approach or chose a traditional coding solution instead.

Organisational problems included:

- Some team members were unable to participate because all the tasks were beyond their technical capabilities. These members became ‘observers’. This left the remaining members (sometimes a minority of the group) to do all the work.

Examples of good practices included:

- Division of a team into sub-teams with sub-team leaders, and separation of team roles into researchers and developers (teams 8, 11).

– Test-driven development (teams 8, 9).
– Metamodel refactoring, to integrate different versions of program metamodels for Java, C# and C++ into a single program metamodel.

Conclusions that can be drawn from this coursework are that an excessively complex task is a bad choice as a first project in MBD, and that students should instead be given the opportunity to build their expertise using less challenging applications. A small percentage of students seem to have a natural aptitude for modelling, others can be trained in modelling skills, and some students seem unable to learn these. Only four teams managed to master the development approach, others either reverted to manual coding or produced incomplete solutions. The total effort expended by successful MBD teams was not in excess of that expended by the successful manual coding team, which suggests that the students on the successful MBD teams were able to attain effective knowledge of MBD.

4 Coursework 2014-15: Electronic health records (EHR) analysis and migration

This project was the OSD assessed coursework for 2014. It was intended to be somewhat easier than the 2013 coursework. Approximately 140 second year undergraduate students participated, divided into 14 teams of 9 or 10 members.

There were three top level use cases: (1) to analyse a dataset of GP patient data conforming to the EHR model of Figure 2 for cases of missing names, address, etc, feature values; (2) to display information on referrals and consultations in date-sorted order; (3) to integrate the GP patient data with hospital patient data conforming to the EHR model of Figure 3 to produce an integrated dataset conforming to a third EHR model (gpmm3). Unlike the 2013 coursework, the initial tasks were intended as relatively simple problems which would help students to gain understanding of the process and tools before proceeding to more challenging tasks. In addition, the coursework only involved working at the domain model (EHR) and instance model (data) level, instead of at all three modelling levels including the metamodel level.

Table 2 summarises the use cases and their subtasks.

4.1 Outcomes

Of the 14 teams, 13 successfully applied the tools and an agile methodology to produce a working solution. Table 3 shows the outcome of the coursework.

Typically the teams divided into subteams, with each subteam given a particular task to develop, so that a degree of parallel development could occur, taking advantage of the independence of the three use cases. Most groups had a defined leader role (this had been advised in the coursework description), and the lack of a leader generally resulted in a poor outcome (as in teams 1, 4, 9, 12, 14).

The key technical difficulties encountered by most teams were:

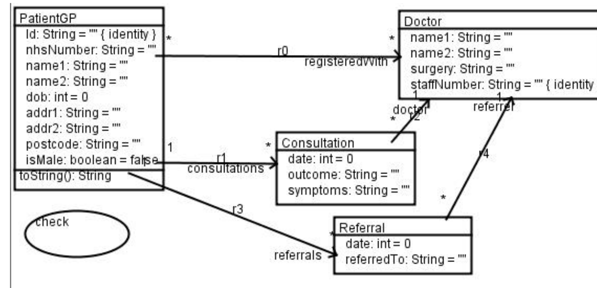

Fig. 2. GP patient EHR model gpmm1

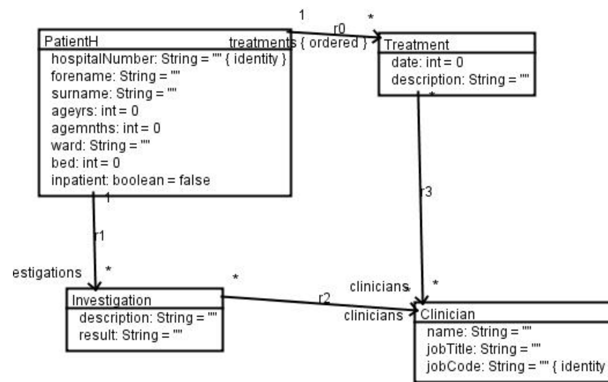

Fig. 3. Hospital patient EHR model gpmm2

Use case	*Subtasks*	*EHR models*
1. Analyse data	1a. Detect missing data in GP dataset	gpmm1
	1b. Detect duplicate patient records	gpmm1
2. View data	2a. Display consultations of each GP patient, in date order	gpmm1
	2b. Display referrals of each GP patient, in date order	gpmm1
3. Integrate data	Combine gpmm1, gpmm2 data into gpmm3	gpmm1, gpmm2 gpmm3

Table 2. Use cases for EHR analysis/migration application

Teams	*Mark range*	*Result*
8, 11	90+	Comprehensive solution and testing, used lead developers and small team modelling
2, 3, 5, 6, 7, 10, 13	80+	High quality solution, used at least one of lead developer/small team practices
1, 4, 9	70-80	Some errors/incompleteness, poorly-organised teams
12	50-60	Failed to complete some tasks. Group split into 2.
14	Below 40	Failed all tasks, group failed to organise.

Table 3. Coursework 2 results

- Lack of prior experience in using UML.
- The unfamiliar style of UML-RSDS compared to tools such as Visual Studio, Net Beans and other IDEs.
- Conceptual difficulty with the idea of MBD.
- Inadequate user documentation for the tools – in particular students struggled to understand how the tools were supposed to be used, and the connection between the specifications written in the tool and the code produced.

As with the first coursework, only a small subset of students expressed enthusiasm for the development approach. The subset of students who were unable to understand the approach was smaller (perhaps 10%) than for the first coursework.

Organisational problems included:

- Team management and communication problems due to the size of the teams and variation in skill levels and commitment within a team.
- Disputes within a team over the approach to take – in the worst case the team split into two.

The problems were generally of a less severe nature than for the previous coursework, and in 12 of 14 cases the student teams overcame these problems. Two teams (12 and 14) had specific management problems, resulting in failure in the case of team 14.

Particular issues can be seen in the following quotes from the team reports:

> "Also a better understanding of UML-RSDS right from the beginning would have been a real plus in solving the tasks" (Team 7)

> "One of the main issues we faced was having to compromise with the limited documentation ... We overcame this issue by thoroughly going through the given documentation, trying and testing many different implementations to solve the tasks. Through trial and error we began to

familiarise ourselves with using the feature-rich UML-RSDS software, ultimately becoming comfortable with using the software in development." (Team 3)

"As with all software there was a learning curve involved in its use, and once we had progressed along this curve and gained some familiarity we found that the software was much easier to use, and every increase in our fluency with the software empowered us to produce higher quality solutions with increasing ease." (Team 8)

"We found however, that becoming more familiar with UML-RSDS was more of a priority in order to be able to solve further tasks, as it is quite different to anything we had used before." (Team 10)

"Many group members did not know UML and had to learn it." (Team 12)

The teams were almost unanimous in identifying that they should have committed more time at the start of the project to understand the tools and the MBD approach. This is a case where the agile principle of starting development as soon as possible needs to be tempered by the need for adequate understanding of a new tool and development technique.

Factors which seemed particularly important in overcoming problems with UML-RSDS and MBD were:

– The use of 'lead developers': a few team members who take the lead in mastering the tool/MBD concepts and who then train their colleagues. This spreads knowledge faster and more effectively than all team individuals trying to learn the material independently. Teams that used this approach had a low training time of 1 week, and achieved an average technical score of 8.66 out of 10, versus 7.18 for other teams. This difference is statistically significant at the 4% level (removing team 14 from the data).
– Pair-based or small team modelling, with subteams of 2 to 4 people working around one machine. This seems to help to identify errors in modelling which individual developers may make, and additionally, if there is a lead developer in each sub-team, to propagate tool and MBD expertise. Teams using this approach achieved an average technical score of 8.25, compared to 7.2 for other teams. This difference is however not statistically significant if team 14 is excluded.

Teams using both approaches achieved an average technical score of 9, compared to those using just one (8.2) or none (6.9).

Another good practice was the use of model refactoring to improve an initial solution with too complex or too finely-divided use cases into a solution with more appropriate use cases.

The impact of poor team management and the lack of a defined process seems more significant for the outcome of a team, compared to technical problems. The Pearson correlation coefficient of the management/process mark of

the project teams with their overall mark is 0.91, suggesting a strong positive relation between team management quality and overall project quality. Groups with a well-defined process and team organisation were able to overcome technical problems more effectively than those with poor management. Groups 3, 5, 7, 11 and 13 are the instances of the first category, and these groups achieved an average of 8.4/10 in the technical score, whilst groups 1, 4, 9, 12 and 14 are the instances of the second category, and these groups achieved an average of 5.8/10 in the technical score.

The outcomes of this coursework were better than for the first coursework: the average mark was 79% in coursework 2, compared to 67.5% for coursework 1. This appears to be due to three main factors: (i) a simpler project involving reduced domain research and technical requirements compared to coursework 1; (ii) improvements to the UML-RSDS tools; (iii) stronger advice to follow an agile development approach; (iv) simpler initial tasks leading on to more complex tasks.

5 Lessons learnt

The courseworks were similar in their assessment procedures and measures, and the educational standard of the students was similar. The scale of the coursework problems, and the level of support were very similar. Thus we consider that their results can be meaningfully compared. Table 4 compares the courseworks with regards to scope and outcomes.

Aspect	*Coursework 2013-14*	*Coursework 2014-15*
Scope of Problem	Text-to-model, model migration, model-to-text	Model analysis, migration, integration
Main organisational difficulties	Some team members unable to participate	Lack of team leadership; internal team disputes
Main technical difficulties	Unable to understand problem or use tools	Time taken to learn UML-RSDS
Successful Practices	Sub-teams; TDD; Metamodel refactoring	Lead developers; Small team modelling
Average mark	67.5%	79%

Table 4. Comparison of courseworks

The results of these two courseworks have identified a number of obstacles in the teaching of practical MBD techniques:

– Students need to gain knowledge in several different technical areas in order to complete the courseworks: domain-specific knowledge about the particular

problem; knowledge of the meaning of OCL constraints as specifications of transformations; tool expertise in UML-RSDS. The conceptual challenge of working with multiple languages at different levels of abstraction (UML, metamodels of source and target languages, instance models) was a barrier for some students.
- Students need to organise themselves effectively in teams, and to work successfully with people who they may not know, and who have a wide range of skill levels and abilities. Conflicts between team members need to be resolved.

On the other hand, these aspects make the coursework a realistic exercise in industrial software practice: where developers are often expected to acquire expertise in multiple new areas and tools, and must be able to work effectively with others.

Deficiencies with the UML-RSDS tools were a significant hinderance to students: the lack of a syntax-aware editor, and the lack of feedback on syntax or type errors. Other MBD technologies were considered as alternatives to UML-RSDS, specifically Eclipse and iUML. However these were rejected because their complexity was higher than for UML-RSDS, which only requires the use of simple file-system organisation skills and the use of a single tool and single tool interface and modelling language in order to carry out all MBD stages.

One positive outcome of the courseworks was that many students gained stronger knowledge and understanding of UML class diagrams and OCL constraints through the practical use of these notations on a significant problem. The quote above from team 12 shows that despite having been taught theoretical material in UML, some students felt that they did not know the language when they started the coursework. Students also gained deeper knowledge of agile approaches by putting into practice agile development techniques.

Recommendations that can be made based on our experience are:

1. Introduce software engineering and UML concepts in first year courses, so that students have more background in the ideas before starting the course. Previously, a software engineering course had been included in the first year of the CS programme, however by 2013 it had been discontinued because it was felt that students need to work on large projects before they can understand the motivation for software engineering. We consider instead that SE and modelling principles should be taught alongside programming, otherwise there is the tendency for students to become fixed in the "only code matters" mindset and to regard courses such as OSD as irrelevant.
2. Include more material in OSD on metamodelling and different modelling levels, to clarify these issues before the coursework.
3. Provide stronger guidance on team working and management: emphasise the need for collaboration, communication and coordination. Some teams split apart because of their inability to reach agreement on the approach/solution to be adopted. In some cases (for Coursework 2 particularly) the more highly-skilled developers attempted to solve the coursework by themselves and excluded other team members from involvement.

The skill levels and aptitudes of team members should be taken into account, so that individuals who cannot learn modelling nonetheless have useful tasks to perform, such as testing, research, or writing the report.
4. Formalise the concept of lead developers, and provide specific MBD and UML-RSDS training to two or three self-selected members from each group, in the first week of the project, so that they can perform this role in their teams.
5. Recommend small team modelling to the groups.
6. Improve motivation by choosing a more interesting problem or domain.

We will put the last five recommendations into practice for the 2015-16 coursework and evaluate any effect these changes have on the coursework outcomes. In addition, the UML-RSDS tools will be improved with a help facility and better documentation.

6 Related work

Surveys of the industrial use of MBD [2, 5–7] have identified several obstacles to the adoption of an MBD approach:

1. Lack of process models for MBD.
2. The need for developer proficiency in complex MBD tools.
3. The need to re-organise development teams to include new roles, such as domain experts and modelling experts.
4. The need to adopt a changed software development lifecycle.
5. Immature and non-interoperable tools.

The same problems were also encountered in our projects, and were compounded by the inexperience of the students compared to professional developers. By recommending that the teams adopt an agile process such as Scrum, we aimed to reduce the first and fourth obstacles. The use of lead developers helps to reduce the second obstacle.

A fundamental problem which remains with MBD is that it appears that only a minority of developers have the ability to work at the modelling level. Thus, in practice, a development team would include an MBD team working as part of a larger conventional team, and there is the need for seamless integration of MBD-produced code and manually-produced code.

The study of [1] considers the use of MBD tools in an educational setting. Their example coursework also involves a complete MBD life-cycle from abstract models to executable implementations. The coursework is substantially more complex than ours, and is a term-length (16 weeks) project on an elective advanced-level module with a small cohort (24 students). About half of the students were already working in the software industry. Teams were between four and five students. Eclipse EMF was used in the project, and the students found this difficult to use, encountering similar problems regarding documentation, tool process and error reporting that we found with students trying to use

UML-RSDS in our course. Team management problems are not considered in [1]. The feedback from the coursework was generally positive, with students identifying that the project made them more aware of the importance of modelling. Our own courseworks were in a more challenging environment, of a compulsory second year course, with a much larger cohort of students, and with larger teams, carried out in a shorter period. Therefore, team management, motivation and training issues are more evident in our projects, and we consider that these issues need to be addressed both for educational projects and for industrial use of MBD, along with the more obvious problems of poor quality tool support and conceptual understanding difficulties.

7 Conclusions

We have analysed the process and outcomes of two assessed problems in the practical application of MBD and agile development, involving a total of over 250 students. From these we have identified lessons learnt and guidelines for the future practical teaching of agile MBD. We have also compared our experiences with other published analyses of MBD projects.

References

1. P. Clarke, Y. Wu, A. Allen, T. King, *Experiences of teaching model-driven engineering in a software design course*, Computer Science Education, vol. 21, issue 4, 2011.
2. J. Hutchinson, J. Whittle, M. Rouncefield, S. Kristoffersen, *Emperical assessment of MDE in industry*, ICSE 11, ACM, 2011.
3. K. Lano, *The UML-RSDS manual*, http://www.dcs.kcl.ac.uk/staff/kcl/umlrsds.pdf, 2015.
4. M. B. Nakicenovic, *An Agile Driven Architecture Modernization to a Model-Driven Development Solution*, International Journal on Advances in Software, vol 5, nos. 3, 4, 2012, pp. 308–322.
5. B. Selic, *What will it take? A view on adoption of model-based methods in practice*, Software systems modeling, 11: 513–526, 2012.
6. S. Stavru, I. Krasteva, S. Ilieva, *Challenges of model-driven modernization: an agile perspective*, MODELSWARD 2013, pp. 219–230.
7. J. Whittle, J. Hutchinson, M. Roucefield, *The state of practice in Model-driven Engineering*, IEEE Software, May/June 2014, pp. 79–85.