Improving the Application of Agile Model-based Development: Experiences from Case Studies

K. Lano
H. Alfraihi
S. Yassipour-Tehrani
Dept. of Informatics
King's College London
London, UK
Email: kevin.lano@kcl.ac.uk, hessa.alfraihi@kcl.ac.uk,
s.yassipour-tehrani@kcl.ac.uk

H. Haughton
Holistic Risk Solutions Ltd
Croydon, UK

Email: howard_haughton@btinternet.com

***Abstract*—Agile model-based development has the potential to combine the benefits of both agile and model-based development (MBD) approaches: rapid automated software generation, lightweight development processes and direct customer involvement. In this paper, we analyse three application case studies of agile MBD, and we identify the factors which have contributed to the success or failure of these applications. We propose an improved agile MBD approach, and give guidelines on its application, in order to increase the effectiveness and success rate of applications of agile MBD.**

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

I. INTRODUCTION

Agile development and model-based development (MBD) are two alternative software development approaches which have been devised to address the 'software crisis' of software project failures and excessive development costs. Both approaches have been adopted by industry to a certain extent, and with some evidence of success. But both approaches also have drawbacks and limitations, which have restricted their uptake.

The idea of combining the approaches into an 'agile MBD' approach has been explored, with the intention that such an approach would avoid the deficiencies of the individual methods [5][12]. In some ways, agile and MBD development approaches are compatible and complementary. For example:

- Both agile development and MBD aim to reduce the gap between requirements analysis and implementation, and hence the errors that arise from incorrect interpretation or formulation of requirements. Agile development reduces the gap by using short incremental cycles of development, and by direct involvement of the customer during development, whilst MBD reduces the gap by automating development steps.
- Executable models (or models from which code can be automatically generated) of MBD potentially serve as a good communication medium between developers and stakeholders, supporting the collaboration which is a key element of agile development.
- Automated code generation accelerates development, in principle, by avoiding the need for much detailed manual low-level coding.
- The need to produce separate documentation is reduced or eliminated, since the executable model is its own documentation.

On the other hand, the culture of agile development is heavily code-centric, and time pressures may result in fixes and corrections being applied directly to generated code, rather than via a reworking of the models, so that models and code become divergent. A possible corrective to this tendency is to view the reworking of the model to align it to the code as a necessary 'refactoring' activity to be performed as soon as time permits. We have followed this approach in several time-critical MBD applications.

Tables I and II summarise the parallels and conflicts between MBD and Agile development.

TABLE I. ADAPTIONS OF AGILE DEVELOPMENT PRACTICES FOR MBD

Practice	*Adaption*
Refactoring for quality improvement	Use model refactoring, not code refactoring
Test-based validation	(i) Generate tests from models (ii) Correct-by-construction code generation
Rapid iterations of development	Rapid iterations of modeling + Automated code generation
No documentation separate from code	Models are both code and documentation

TABLE II. CONFLICTS BETWEEN AGILE DEVELOPMENT AND MBD

Conflict	*Resolutions*
Agile is oriented to source code, not models	(i) Models as code (ii) Round-trip engineering (iii) Manual re-alignment
Agile focus on writing software, not documentation	Models are both documentation and software
Agile's focus on users involvement in development, versus MBD focus on automation	Active involvement of users in conceptual and system modelling

There are therefore different ways in which agile development and MBD can be combined, and the current agile MBD methods adopt different approaches for this integration. In this paper, we examine one possible approach for combining

agile and MBD, based on the Unified Modeling Language Rigorous Specification, Design and Synthesis (UML-RSDS) formalism and tools [9], which we summarise in Section II. This is compared with other agile MBD approaches in Section III. We then report results from three case studies using the UML-RSDS approach (Sections IV,V,VI), and in Section VII, we summarise the lessons learnt from these applications and give guidelines for improving the approach. Section VIII gives conclusions.

II. UML-RSDS

UML-RSDS is based on the class diagram, use case and Object Constraint Language (OCL) notations of UML. System specifications can be written in these notations, and then a design expressed using UML activities can be automatically synthesised from the specifications. Finally, executable code in several alternative languages (Java, C# and C++) can be automatically synthesised from the design [8]. Both structural and behavioural code is synthesised, and a complete executable is produced. The aim of the approach is to automate code production as much as possible, including code optimisation, so that system specifications can be used as the focus of development activities. Some configuration of the design choices can be carried out manually. The system construction process supported by UML-RSDS is shown in Figure 1.

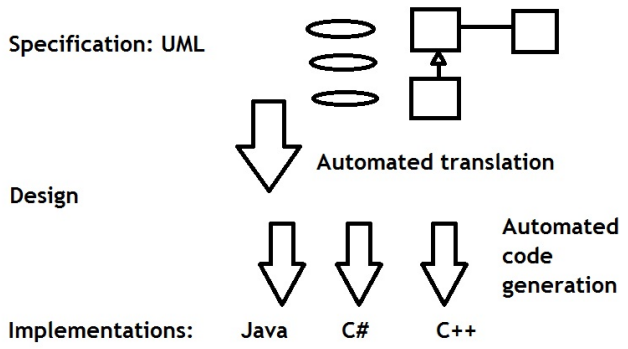

Figure 1. UML-RSDS software production process

An example specification of behaviour in UML-RSDS, from case study 2, is the following use case postcondition, which checks the GP data for duplicated patients (task 1b):

```
p : PatientGP & Id < p.Id &
name1 = p.name1 & name2 = p.name2 &
dob = p.dob & isMale = p.isMale   =>
    ("Patients " + self + " and " + p +
     " seem to be duplicates")->display()
```

This iterates over *self* : *PatientGP*, and displays a warning message for each other patient *p* that has the same name, date of birth and gender as *self*, but a different id value.

The UML-RSDS approach supports agile development, with the options (ii) (correct-by-construction code generation) and (i) (models as code) from Tables I and II being used to combine MBD and agile concepts.

III. Related work

A small number of other agile MBD approaches have been formulated and applied: Executable UML (xUML) [13]; Sage [7]; MDD System Level Agile Process (MDD-SLAP) [18]; Hybrid MDD [4]. Both xUML and UML-RSDS use the principle that "The model is the code", and support incremental system changes via changes to the specification. There is a clearly-defined process for incremental revision in UML-RSDS, MDD-SLAP and Hybrid MDD. MDD-SLAP and Hybrid MDD define explicit integration processes for combining synthesised and hand-crafted code.

Explicit verification processes are omitted from Sage and Hybrid MDD. In MDD-SLAP, simulation and test-driven modelling is used for validation and verification [18]. Some support for formal validation and verification is provided by xUML and UML-RSDS: the iUML tool for xUML has support for simulation, and UML-RSDS provides correctness analysis via a translation to the B formal method. By automating code generation, agile MBD approaches should improve the reliability and correctness of code compared to manual-coding development. All the approaches are focussed on one-way forward engineering, and do not support round-trip engineering, which means that synchronisation of divergent code and models is a manual process.

UML-RSDS and xUML are based on modelling using the standard UML model notations, with some variations (action language in the case of xUML, use cases specified by constraints in UML-RSDS), and on following a general MDA process: Computation-independent Model (CIM) to Platform-independent Model (PIM) to Platform-specific Model (PSM) to code. Platform modelling is explicitly carried out in xUML but not in UML-RSDS, which restricts developers to Java-like languages for the executable code. Sage uses variants of UML models oriented to reactive system definition using classes and agents. These include environmental, design, behavioural and runtime models. An executable system is produced by integration of these models. MDD-SLAP maps MDD process activities (requirements analysis and high-level design; detailed design and code generation; integration and testing) into three successive sprints used to produce a new model-based increment of a system. Hybrid MDD envisages three separate teams operating in parallel: an agile development team hand-crafting parts of each release; a business analyst team providing system requirements and working with a MDD team to produce domain models. The MDD team also develops synthesised code. MDD-SLAP and Hybrid MDD have the most elaborated development processes. The survey of [5] identifies that Scrum-based approaches such as MDD-SLAP are the most common in practical use of agile MBD (5 of the seven cases examined), with XP also often used (4 of 7 cases). The agile MDD approach in the case of [15] used Scrum and Kanban.

IV. Case study 1: FIXML code generation

This case study was based on the problem described in [15]. Financial transactions can be electronically expressed using formats, such as the Financial Information eXchange (FIX) 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. In [15], the author proposed to address this problem by automatically synthesising

program code representing the transaction messages from a single XML definition of the message format, so that users would always have the latest code definitions available. For this case study we restricted attention to generating Java, C# and C++ class declarations from messages in FIXML 4.4 format [2][3].

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.

In principle, these use cases could be developed independently, although the subteams or developers responsible for use cases 2 and 3 need to agree on the programming language metamodel(s) to be used.

The problem was set as the assessed coursework (counting for 15% of the course marks) for the second year undergraduate course "Object-oriented Specification and Design" (OSD) at King's College in 2013. It was scheduled in the last four weeks at the end of the course. OSD covers UML and MBD and agile development at an introductory level. Students also have experience of team working on the concurrent Software Engineering Group project (SEG). Approximately 120 students were on the course, and these were divided into 12 teams of 10 students each, using random allocation of students to teams.

The students were instructed to use UML-RSDS to develop the three use cases of the problem, by writing specifications using the UML-RSDS tools and generating code from these specifications. They were also required to write a team report to describe the process they followed, their team organisation, and the system specification. The case study involves research into FIXML, XML, UML-RSDS and C# and C++, and carrying out the definition of use cases in UML-RSDS using OCL. 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, and links to the UML-RSDS tools and manual. The XML metamodel was provided, and a UML-RSDS library to parse XML was also given to the students to use. Each week there was a one hour timetabled lab session where teams could meet and ask for help from postgraduate students who had some UML-RSDS knowledge. The outcome of the case study is summarised in Table III.

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).

TABLE III. CASE 1 RESULTS

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.

- Metamodel refactoring, to integrate different versions of program metamodels for Java, C# and C++ into a single program metamodel.

Exploratory and evolutionary prototyping were used by most teams as their main development process. However, most teams experienced substantial obstacles in the project, due to (i) problems with the interface of the UML-RSDS tools, which did not conform to the usual style of development environment (such as NetBeans) which the students were familiar with; (ii) problems understanding and using the MBD executable model concept. 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 approach can be feasible even in adverse circumstances.

V. Case study 2: Electronic health records (EHR) analysis and migration

This case study 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. Students were allocated randomly to teams.

There were three required use cases: (1) to analyse a dataset of GP patient data conforming to the class diagram of Figure 2 for cases of missing names, address, or other 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 class diagram of Figure 3 to produce an integrated dataset conforming to a third class diagram (gpmm3).

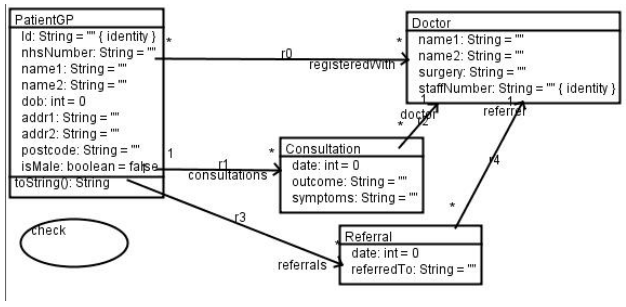

Figure 2. GP patient model gpmm1

Table IV summarises the use cases and their subtasks.

As with case study 1, the teams were required to use UML-RSDS to develop the system, and to record their organisation and results in a report. Teams were advised to select a leader,

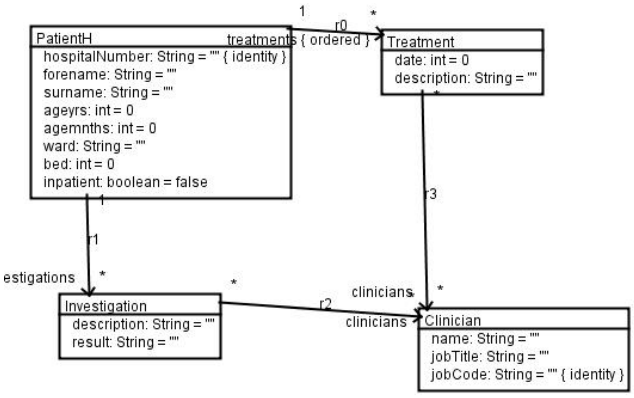

Figure 3. Hospital patient model gpmm2

TABLE IV. USE CASES FOR EHR ANALYSIS/MIGRATION

Use case	*Subtasks*	*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 V. OUTCOMES OF EHR CASE STUDY

Team	*Training time*	*Technical outcome*	*Agile process*	*Activities, issues, process*
1	$>$ 1 week	8/10	8/10	Disorganised and individual working
2	1 week	9/10	8/10	No experience of large teams
3	$>$ 1 week	8/10	9/10	Used pair modelling, proactive time planning
4	1 week	7/10	8/10	No leader. Parallel working
5	1 week	9/10	8/10	Lead developers
6	1 week	8/10	9/10	Used Scrum, subteam modelling, model refactoring
7	1 week	8/10	9/10	Risk analysis, paired modelling
8	1 week	9/10	9/10	Small team modelling. Lead developers trained team
9	$>$ 1 week	7/10	7/10	No leader, disorganised
10	1 week	8/10	8/10	Detailed planning, scheduling. Lead developers trained team
11	1 week	9/10	9/10	Used XP
12	$>$ 1 week	7/10	5/10	Team split into 2
13	2 weeks	8/10	8/10	Strong leadership
14	2 weeks	0/10	0/10	Failed to work as a team

and to apply an agile development process, although a specific process was not mandated. A short (2 page) requirements document was provided, and links to the UML-RSDS tools and manual. The three EHR models were provided. Each week there was a one hour timetabled lab session where teams could meet and ask for help from postgraduate students who had some UML-RSDS knowledge.

A. Outcomes

Of the 14 teams, 13 successfully applied the tools and an agile methodology to produce a working solution. Table V shows the characteristics of the different team solutions. Training time refers to the time needed to learn MBD using UML-RSDS.

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). As with case study one, exploratory and evolutionary prototyping of specifications was used by the teams.

The key difficulties encountered by most teams were:

- Lack of prior experience in using UML.
- The unfamiliar style of UML-RSDS compared to tools such as Visual Studio, Net Beans and other development environments.
- Conceptual difficulty with the idea of MBD and the use of OCL to specify system functionality.
- 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.
- Team management and communication problems due to the size of the teams and variation in skill levels and commitment within a team.

Nonetheless, in 12 of 14 cases the student teams overcame these problems. Two teams (12 and 14) had severe management problems, resulting in failure in the case of team 14.

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, 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. An agile process seems to be helpful in achieving a good technical outcome: the correlation of the agile process and technical outcome scores in Table V is 0.93.

The outcomes of this case study were better than for the first case study: the average mark was 79% in case study 2, compared to 67.5% for case study 1. This appears to be due to three main factors: (i) a simpler case study involving reduced domain research and technical requirements compared to case study 1. In particular there was no need to understand and use an external library such as the XML parser; (ii) improvements to the UML-RSDS tools; (iii) stronger advice to follow an agile development approach.

In conclusion, this case study illustrated the problems which may occur when industrial development teams are introduced to MBD and MBD tools for the first time. The positive conclusions which can be drawn are that UML-RSDS appears to be an approach which quite inexperienced developers can use successfully for a range of tasks, even with limited access to tool experts, and that the difficulties involved in learning the tools and development approach are not significantly greater than those that could be encountered with any new SE environment or tools.

VI. Case study 3: Collateralized Debt Obligations risk estimation

This case study concerns the risk evaluation of multiple-share financial investments known as *Collateralized Debt Obligations* (CDO), where a portfolio of investments is partitioned into a collection of sectors, and there is the possibility of contagion of defaults between different companies in the same sector [1][6]. Risk analysis of a CDO contract involves computing the overall probability $P(S = s)$ of a financial loss s based upon the probability of individual company defaults and the probability of default infection within sectors.

Both a precise (but very computationally expensive) and an approximate version of the loss estimation function $P(S = s)$ were required. The case study was carried out in conjunction with a financial risk analyst, who was also the customer of the development. Implementations in Java, C# and C++ were required.

The required use cases and subtasks are given in Table VI. Use case 3 depends upon tasks 2a and 2b of use case 2. Unlike case studies 1 and 2, team management was not a problem because this was a single-developer project. In addition the developer was an expert in UML-RSDS. Therefore the focus of interest in this case study is how effectively agile development with UML-RSDS can be used for this domain.

TABLE VI. USE CASES FOR CDO RISK ANALYSIS

Use case	*Subtasks*	*Description*
1. Load data		Read data from a .csv spreadsheet
2. Calculate Poisson approximation of loss function	2a Calculate probability of no contagion	
	2b. Calculate probability of contagion	
	2c. Combine 2a, 2b	
3. Calculate precise loss function		
4. Write data		Write data to a .csv spreadsheet

First, a phase of research was needed to understand the problem and to clarify the actual computations required. Then tasks 2a, 2b and 2c were carried out in a first development iteration, as these were considered more critical than use cases 1 or 4. Exploratory and evolutionary prototyping were used. Then, the use case 3 was performed in development iteration 2, and finally use cases 1 and 4 – which both involved use of manual coding – were scheduled to be completed in a third development iteration. A further external requirement was introduced prior to this iteration: to handle the case of cross-sector contagion. This requirement was then scheduled in the third iteration, and tasks 1 and 4 in a fourth iteration.

Figure 4 shows the class diagram of the solution produced at the end of the first development iteration.

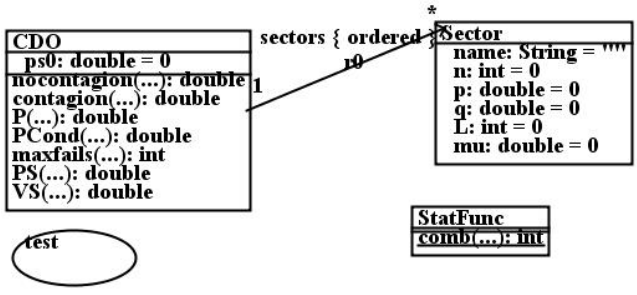

Figure 4. CDO version 1 system specification

The following agile development techniques were employed:

- Refactoring: the solutions of 2a and 2b were initially expressed as operations *nocontagion*, *contagion* of the CDO class (Figure 4). It was then realised that they would be simpler and more efficient if defined as Sector operations. The refactoring Move Operation was used. This refactoring did not affect the external interface of the system.
- Customer collaboration in development: the risk analyst gave detailed feedback on the generated code as it was produced, and carried out their own tests using data such as the realistic dataset of [6].

It was originally intended to use external hand-coded and optimised implementations of critical functions such as the combinatorial function $comb(int\ n, int\ m)$. However, this would have resulted in the need for multiple versions of these functions to be coded, one for each target implementation language, and would also increase the time needed for system integration. It was found instead that platform-independent specifications could be given in UML-RSDS which were of acceptable efficiency.

The initial efficiency of the approximate solution was too low, with calculation of $P(S = s)$ for all values of $s \leq 20$ on the test data of [6] taking over 2 minutes on a standard Windows 7 laptop. To address this problem, the recursive operations and other operations with high usage were given the stereotype $\ll$ *cached* $\gg$ to avoid unnecessary recomputation. This stereotype means that operations are implemented using the *memoisation* technique of [14] to store previously-computed results. Figure 5 shows the refactored system specification at the end of the third development iteration.

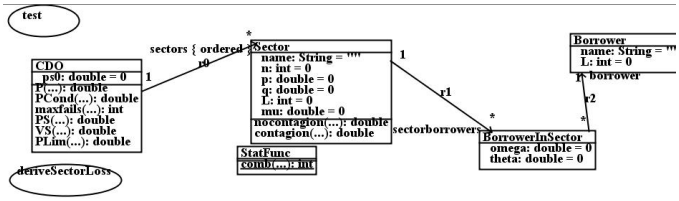

Figure 5. CDO version 3 system specification

Table VII shows the improvements in efficiency which memoisation provides, and the results for generated code in other language versions. The approximate version of $P(S = s)$ is compared.

TABLE VII. EXECUTION TIMES FOR CDO VERSIONS

Version	*Execution time for first 20 $P(S = s)$ calls*	*Execution time for first 50 $P(S = s)$ calls*
Unoptimised Java	121s	–
Optimised Java	32ms	93ms
C#	10ms	20ms
C++	62ms	100ms

Our experiences on this case study illustrate the UML-RSDS principles:

- Optimisation and refactoring should be carried out at the specification level in a platform-independent manner where possible, not at the code level.
- The scope of MBD should be extended as far as possible across the system development, reducing the scope of manual coding and integration wherever possible.

In conclusion, this case study showed that a successful outcome is possible for agile MBD in the highly demanding domain of computationally-intensive financial applications. A generic MBD tool, UML-RSDS, was able to produce code of comparable efficiency to existing hand-coded and highly optimised solutions.

VII. Guidelines for an improved agile MBD process

The case studies have identified the need for a well-defined agile MBD process for using UML-RSDS, and some techniques for improving the adoption and application of UML-RSDS, in addition to necessary technical improvements in the tools. In general it was found that a development approach using exploratory prototyping (of the system specification) at the initial stages, and evolutionary prototyping at later stages, was effective.

The following guidelines for adoption and application of agile MBD are proposed, on the basis of our experiences in the presented and other case studies:

- **Utilise lead developers** When introducing MBD to a team inexperienced in its use, employ a small number of team members – especially those who are most positive about the approach and who have appropriate technical backgrounds – to take the lead in acquiring technical understanding and skill in the MBD approach. The lead developers can then train their colleagues.
- **Use paired or small team modelling** Small teams working together on a task or use case can be very effective, particularly if each team contains a lead developer, who can act as the technical expert. It is suggested in [18] that such teams should also contain a customer representative.
- **Use a clearly defined process and management structure** The development should be based on a well-defined process, such as XP, Scrum, or the MBD adaptions of these given in this paper or by MDD-SLAP and Hybrid MDD. A team leader who operates as a facilitator and co-ordinator is an important factor, the leader should not try to dictate work at a fine-grained level, but instead enable sub-teams to be effective, self-organised and to work together.
- **Refactor at specification level** Refactor models, not code, to improve system quality and efficiency.
- **Extend the scope of MBD** Encompassing more of the system into the automated MBD process reduces development costs and time.

The first three of these are also recommended as good practices for agile development in general [10][11][17].

A detailed agile MBD process for UML-RSDS can be based upon the MDD-SLAP process. Each development iteration is split into three phases (Figure 6):

- **Requirements and specification:** Identify and refine the iteration requirements from the iteration backlog,

and express new/modified functionalities as system use case definitions. Requirements engineering techniques such as exploratory prototyping and scenario analysis can be used. This stage corresponds to the Application requirements sprint in MDD-SLAP. Its outcome is an iteration backlog with clear and detailed requirements for each work item.
If the use of MBD is novel for the majority of developers in the project team, assign lead developers to take the lead in acquiring technical skills in MBD and UML-RSDS.

- **Development, verification, code generation:** Subteams allocate developers to work items and write unit tests for their assigned use cases. Subteams work on their items in development iterations, using techniques such as evolutionary prototyping, in collaboration with stakeholder representatives, to construct detailed use case specifications. Formal verification at the specification level can be used to check critical properties. Reuse opportunities should be regularly considered, along with specification refactoring.
  Daily Scrum-style meetings can be held within subteams to monitor progress, update plans and address problems. Techniques such as a Scrum board and burndown chart can be used to manage work allocation and progress. The phase terminates with the generation of a complete code version incorporating all the required functionalities from the iteration backlog.
- **Integration and testing:** Do regular full builds, testing and integration in an integration iteration, including integration with other software and manually-coded parts of the system.

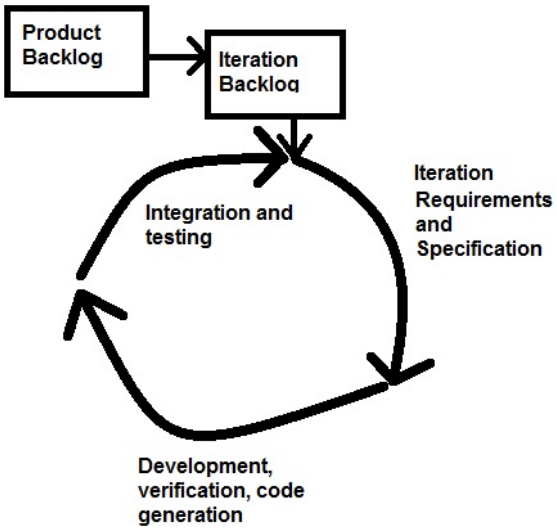

Figure 6. UML-RSDS process

VIII. Conclusions

We have analysed the process and outcomes of three case studies of MBD and agile development, involving a total of over 250 developers. From these cases, we have identified guidelines for the use of agile MBD, and an improved agile MBD process for UML-RSDS. In future work, we will develop a systematic evaluation framework for agile MDD application, and investigate extensions of our agile MDD process.

References

[1] M. Davis and V. Lo, "Infectious Defaults", Quantitative Finance, vol. 1, no. 4, 2001, pp. 382–387.

[2] http://fixwiki.org/fixwiki/FPL:FIXML_Syntax. Accessed 11.9.2015.

[3] http://www.fixtradingcommunity.org. Accessed 11.9.2015.

[4] G. Guta, W. Schreiner, and D. Draheim, "A lightweight MDSD process applied in small projects", Proceedings 35th Euromicro conference on Software Engineering and Advanced Applications, IEEE, 2009, pp. 255-258.

[5] S. Hansson, Y. Zhao, and H. Burden, "How MAD are we?: Empirical evidence for model-driven agile development", XM Workshop, MODELS 2014, 2014, pp. 2-11.

[6] O. Hammarlid, "Aggregating sectors in the infectious defaults model", Quantitative Finance, vol. 4, no. 1, 2004, pp. 64–69.

[7] J. Kirby, "Model-driven Agile Development of Reactive Multi-agent Systems", COMPSAC '06, 2006, pp. 297–302.

[8] K. Lano and S. Kolahdouz-Rahimi, "Constraint-based specification of model transformations", Journal of Systems and Software, vol. 88, no. 2, February 2013, pp. 412–436.

[9] K. Lano, The UML-RSDS manual, http://www.dcs.kcl.ac.uk/staff/kcl/umlrsds.pdf, 2015.

[10] L. Lavazza, S. Morasca, D. Taibi, and D. Tosi, "Applying Scrum in an OSS Development Process: an Empirical Evaluation", in 11th International Conference XP 2010, 2010, pp. 147–159.

[11] R. C. Martin, "Agile Software Development: Principles, Patterns and Practices", Prentice Hall, 2003.

[12] R. Matinnejad, "Agile Model Driven Development: an intelligent compromise", 9th International Conference on Software Engineering Research, Management and Applications, 2011, pp. 197–202.

[13] S. Mellor and M. Balcer, "Executable UML: A foundation for model-driven architectures", Addison-Wesley, Boston, 2002.

[14] D. Michie, "Memo functions and machine learning", Nature, vol. 218, 1968, pp. 19–22.

[15] 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.

[16] K. Schwaber and M. Beedble, "Agile software development with Scrum", Pearson, 2012.

[17] D. Taibi, P. Diebold, and C. Lampasona, "Moonlighting Scrum: an agile method for distributed teams with part-time developers working during non-overlapping hours", in ICSEA 2013, pp. 318–323.

[18] Y. Zhang and S. Patel, "Agile model-driven development in practice", IEEE Software, vol. 28, no. 2, 2011, pp. 84–91.