Journal of Computing and Security

July 2019, Volume 6, Number 2 (pp. 43–52)

http://www.jcomsec.org

eXtreme Modeling: an approach to agile model-based development

Shekoufeh Kolahdouz-Rahimi [a,*], Kevin Lano [b], Hessa Alfraihi [c], Howard P. Haughton [d]

[a] *MDSE Research Group, Dept. of Software Engineering, University of Isfahan, Isfahan, Iran.*
[b] *Dept. of Informatics, King's College London, London, UK.*
[c] *Dept. of Information Systems, Princess Nourah bint Abdulrahman University, Saudi Arabia.*
[d] *Holistic Risk Solutions Ltd, Croydon, UK.*

ARTICLE INFO.

Article history:
Received: 5 February 2019
Revised: 14 September 2019
Accepted: 6 October 2019
Published Online: 13 November 2019

Keywords:
Model-Based Development (MBD), Model-Driven Development (MDD), Agile Development.

ABSTRACT

Model-based development (MBD) is the development of software systems using graphical and textual models such as UML class diagrams. MBD and related approaches such as Model-driven development (MDD) have had some success within specific application domains, such as the automotive industry. Agile software development approaches such as Scrum and eXtreme Programming (XP) have been widely adopted in many different industry sectors. These approaches emphasise iterative development and close customer collaboration. eXtreme Modeling (XM) is a model-based development analogue of eXtreme Programming: it is an agile development approach based on the use of software models to specify and synthesise software systems. In this paper we look at the track record of agile and model-based development, and we consider the case for combining these approaches into XM to obtain benefits from both approaches: rapid automated software generation, lightweight development processes, and direct customer involvement. An example application of XM in the financial services domain is described.

1 Introduction

Agile development and model-based development (MBD) are two alternative software development approaches which have been devised to address the 'software crisis' of increasing system complexity and development costs, and the problem of expensive software project failures and overruns.

Agile development aims to trim away superfluous development activities and to focus on the activities which add value. The approach accepts requirements changes as inevitable, and supports such changes by using short development iterations which produce some increment of the system that meets new or modified requirements. To ensure correctness of the delivered system with respect to customer needs, close collaboration and communication with the stakeholders is mandated during the development.

A series of values and principles for agile development have been formulated [1]. The four key values are:

- Responding to change is more important than following a plan.
- Producing working software is more important than comprehensive documentation.
- Individuals and interactions are emphasised over processes and tools.
- Customer collaboration is emphasised over con-

* Corresponding author.
Email addresses: sh.rahimi@eng.ui.ac.ir(Sh.Rahimi), kevin.lano@kcl.ac.uk (K.Lano), haalfraihi@pnu.edu.sa (H.Alfraihi), haughton@btinternet.com (H. Haughton)
https://dx.doi.org/10.22108/jcs.2019.115445.1018
ISSN: 2322-4460

tract negotiation.

There are 12 principles:

(1) Satisfy customers through frequent working software delivery.
(2) Embrace changing requirements even at late stages of development.
(3) Deliver working software frequently in short cycles.
(4) Work daily with business people throughout the project.
(5) Support and trust motivated individuals.
(6) Face-to-face conversation is the ultimate way of conveying information.
(7) Working software is the primary measure of progress.
(8) Promote sustainable development to maintain a constant pace indefinitely.
(9) Constant orientation to technical excellence and good design.
(10) Simplicity is essential.
(11) Promote self-organising teams.
(12) Encourage inception and adaptation.

A number of agile practices have been defined, of which the most popular are daily review meetings, short iterations, and prioritised backlogs [2]. Agile methods include XP [3] and Scrum [4]. These have become widely used in the software industry, with definite evidence of success [5], [6], [2].

However, agile development has also been criticised for its dependence upon highly-skilled developers, and for its failure to consider non-functional requirements [7]. It also focusses upon the development of a specific product in each process, and does not consider reuse possibilities or long-term support for the development of product families [8].

Model-based development may appear to be the diametric opposite of agile development: MBD focusses on building models and (in cases such as the Rational Unified Process or the OMG's Model-driven Architecture, MDA) on an elaborate staged plan-based development process, in contrast to the agile focus on code and minimal processes. MBD aims to address the software crisis by reducing the gap between requirements and implementation, by focussing on the construction of domain-level models, and by automating development steps. In Model-driven development (MDD), models supersede code throughout the development process.

MBD has been successfully adopted in specific industries, most notably in the production of vehicle systems in the automobile industry, where software correctness is a priority [9]. There is evidence that automated software synthesis can increase productivity by large factors [10]. However, significant problems remain with MBD adoption [11], including:

- Costs of training and changes to work practices.
- Difficulties of maintaining consistency between multiple models, and between models and code.
- Poor-quality and immature tool support, lacking sufficient customisability or interoperability.

A combined agile and MBD approach could help to address the problems with the individual agile and MBD approaches, as shown in Table 1.

Table 1. Problems of Agile and MBD Development.

Problem	**Resolutions in Agile MBD**
Agile: Cost of customer meetings	Models are more abstract than code, hence easier and faster to review
Agile: Lack of design and documentation	Models provide documentation
Agile: Does not address product-line development	Recognize reuse opportunities during development, and construct platform/product line models
MBD: Poor-quality tools	Dedicated tooling sub-team
MBD: Disruption to development process	Effective integration of MBD + hand-written components.
MBD: Heavyweight and complex processes	Lean development, only essential documentation
MBD: Divergence between models and code	Work primarily at model level
Both: Require high-skilled developers	Modelling requires fewer resources than coding
Both: Do not address non-functional requirements	Introduce explicit requirements engineering phase in each agile iteration

The idea of combining MBD and agile approaches into an 'agile MBD' approach has been explored by several researchers, and a small number of agile MBD approaches have been formulated and applied: Executable UML (xUML) [12]; Sage [13]; MDD System Level Agile Process (MDD-SLAP) [14], and Hybrid MDD [15]. Hybrid MDD introduced the concept of a tooling (MDD) team working together with agile development and business analysis teams, whilst 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. Both MDD-SLAP and Hybrid MDD define explicit integration processes for combin-

ing synthesised and hand-crafted code. MDD-SLAP has been used in the telecoms sector, whilst Hybrid MDD has been used for web systems development. Support for round-trip engineering and for model-based test generation is a significant omission from existing agile MBD approaches, however.

The survey of [16] 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).

In this paper, we describe an approach "eXtreme Modelling" (XM) for combining agile and MBD, based on the Unified Modelling Language Rigorous Specification, Design and Synthesis (UML-RSDS) formalism and tools [17], and on the best practices for Agile MBD identified by Hybrid MDD and MDD-SLAP. We summarise the principles and practices of our approach in Section 2, describe tool and process support in Section 3, and report results from an industrial case study in Section 4. Section 6 compares XM to other MBD and Agile MBD methods.

2 Principles and Practices of eXtreme Modelling

XM adopts the four values and 12 principles of agile development, with the proviso that principle 7 "Working software is the primary measure of progress" should be altered to "Working software and accurate models are the primary measures of progress".

XM adds the further principles:

(1) The specification *is* the system.
(2) Maintain a system by changing its specification, not the code.
(3) Build the system that is *needed*, not only the required system.
(4) Maximise reuse of existing specifications and systems. Contribute to reuse from your system.
(5) Maximise the scope of platform-independent specification and design.
(6) Minimise the complexity of software models, metamodels and transformations.
(7) Automate code generation and other development steps where possible and beneficial.

The XP practices *planning game*; *small releases*; *metaphor*; *simple design*; *continuous testing*; *continuous integration*; *40-hour work week*; *on-site customer* are also adopted in XM. The practices *collective code ownership* and *coding standards* are altered to apply to models/specification instead of code.

The key additional XM practices are:

(1) Exploratory and evolutionary prototyping using executable specifications.
(2) Paired or small team modelling.
(3) Formal model reviews.
(4) Refactor at the specification level.
(5) Correctness by construction.
(6) Close collaboration with customers.
(7) Daily review meetings.

The extreme modeller hacks specifications, not code. Working at a higher level of abstraction, closer to the stakeholder and domain level, makes development work more productive, improves communication with stakeholders, and increases agility. Platform-independence of the specification supports system evolution, and means that alternative versions of a system for different platforms can be efficiently produced.

2.1 Principles

The specification *is* the system The system requirements specification (SRS) should define precisely the functional requirements of the system. Highly-automated design and code generation from the SRS removes the need to maintain any other more refined or alternative definition of the system – any changes to the system functionality can be expressed by revising the SRS and regenerating the implementation.

Maintain a system by changing its specification, not the code Cutting code is a labour-intensive and expensive activity. Specifications can be more concise than executable code by a factor of 100 times or more, leading to reduced development times and costs if specification writing and revision replaces coding as the main development activity.

Synthesised code merely expresses, in a particular programming language, functionality which should already be clear from the specification. If testing of the code reveals errors or deficiencies in the specification, the developer should change the specification, not the code.

Build the system that is *needed*, not only the required system Requirements engineering should aim to capture implicit as well as explicit requirements: such unstated requirements can be critical for the actual use of the system, and for its failure or success upon deployment. Agile development has emphasised the importance of direct communication with stakeholder representatives – this needs to be extended to include end users where possible, in order to take their needs into account. Techniques such as ethnographic studies and workplace observation can be used to capture implicit requirements.

Maximise reuse of existing specifications and systems. Contribute to reuse from your system Dur-

ing the requirements phase of an iteration, the developers should identify if existing components can be used to support new requirements. In addition, during the development phase, components which have potential for future reuse should be identified where possible. The recognition of reusable components within a product line may result in the establishment of a substantial library of components to support the development of related products. This can be managed as a *product line platform* for the domain of systems in the product line.

Maximise the scope of platform-independent specification and design The scope of MBD should be extended as far as possible across the system development, to maximise automated code generation, and to reduce the extent of manual coding and integration wherever possible.

Minimise complexity of models, metamodels and transformations Extending agile principle 10 "Simplicity is essential", the MBD artifacts used in XM should emphasise simplicity and avoid unnecessary complexity. Initial models can be sketches used to explore ideas. Models used for artifact generation should also be as simple as possible, with tools used to infer design and code details from high-level specification models.

Following agile principle 9, quality guidelines for model, metamodel and transformation construction should be followed, such as [18] for metamodels, and [19] for QVT-O transformations.

Automate code generation and development whenever possible and beneficial In order to raise productivity and reduce development time, automation of routine development work should be used. Code generators can be used to produce code from models, and it is also possible to synthesise transformations from metamodels [20] and metamodels from models [21].

2.2 Practices

Exploratory and evolutionary prototyping using executable specifications Exploratory prototyping can be used during the requirements phase of an iteration, to clarify and agree the meaning of requirements between the customer and developers. During the development phase, evolutionary prototyping is used to progressively extend the SRS for iteration work items until they fulfil the work item requirements. Subsequently, evolutionary prototyping can be used to optimise the SRS.

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 customer representative, a business analyst and one or more technical MBD experts.

Formal model reviews Both the writer of a specification, and other team members, should regularly review the specification with respect to the requirements and its internal correctness and quality. Opportunities for modularisation and the use of specification patterns [22] should be considered.

Refactor at the specification level Refactor and optimise models, not code, to improve system quality and efficiency. Model improvements and optimisations then take effect for any synthesised code version.

Correctness by construction We don't expect, when we apply a C++ compiler to generate assembly code from a C++ program, to have to verify that the assembly code actually *is* correct with respect to the program text. In the same way, code synthesis from specifications should be so reliable that no effort needs to be wasted in post-hoc verification.

Close collaboration with customers Customers and other stakeholders should be involved in the modeling and specification activities, and in the construction and review of tests, wherever possible.

Daily review meetings A daily Scrum-style meeting can be used to identify progress issues and problems that need to be resolved.

3 UML-RSDS Tools and Process

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#, C++, C, Python) can be automatically synthesised from the design [23]. 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.

The UML-RSDS tools support XM, and provide correct-by-construction code generation. In contrast to other MBD tools, UML-RSDS emphasises simplicity: it uses a single UML model (class diagrams with use cases) instead of multiple models, thus removing the overhead of maintaining model consistency. It uses

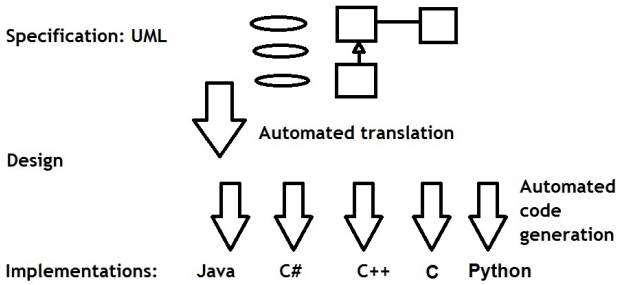

Figure 1. UML-RSDS Software Production Process.

a simplified UML and OCL notation, and a higher level of abstraction compared to approaches such as xUML [12] or FUML [24] – constraints, instead of detailed action language code – thus providing a more concise and clearer specification, closer to the informal business rules and the requirements of the system. The UML-RSDS tools provide support for extensions to define new code generators, and also provide facilities for interoperation with Eclipse. Synthesised code modules may be used as either clients or suppliers relative to hand-crafted or external code modules.

A detailed XM process for UML-RSDS has been defined, adopting ideas from the XP, Scrum, MDD-SLAP and Hybrid MDD processes. Release planning, and the derivation of the iteration backlog from the product backlog is carried out in a standard way, as in Scrum or MDD-SLAP. The Scrum roles of *product owner*, *development team* and development organiser (cf. *Scrum master*) are adopted. Each iteration (sprint) is split into three phases (Figure 2):

- **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, interviews with stakeholders, and scenario analysis can be used. This phase 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 who will initially acquire technical skills in MBD and UML-RSDS, and then subsequently train other team members.
- **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 the development phase, 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, review plans and identify 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 complete code for all the work items from the iteration backlog.
- **Integration and testing:** Do regular full builds, testing and integration in an integration phase, including integration with other software and manually-coded parts of the system.

After each sprint, a sprint review is held, and planning for the next sprint is carried out, identifying the sprint backlog from the updated product backlog.

In parallel with the main development iterations, a tooling team may be employed to provide necessary extensions or adaptions to UML-RSDS. For example, to write new data converters or code generators needed by the iteration. The tooling team also uses XM. A developer from the main iteration acts as the 'customer' for the tooling team work.

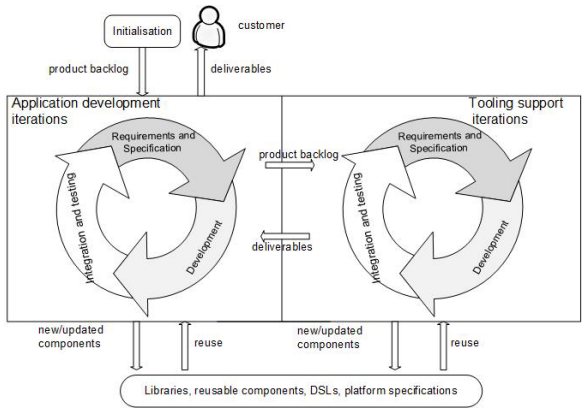

Figure 2. XM Process for Iterations.

A wide range of case studies have been carried out using UML-RSDS [23, 25–27]. The tools are also provided as part of the Eclipse Agile UML toolset (https://projects.eclipse.org/projects/modeling.agileuml).

In the following section, we describe how XM with UML-RSDS has been applied in practice to a system development in the financial services industry.

4 Yield Curve Estimation

This case study consisted of the development of algorithms to estimate yield curves, using the Nelson-

Siegel-Svensson model [28] for such curves and variations on this model.

A *yield curve* defines how interest rates/yields for investments of different maturities/terms varies with the term. The yield is the annual rate of return on an investment. Longer loans/investments tend to have higher yields, so a curve normally increases from low terms to higher terms. The yield $y(t)$ for investments of duration t is also referred to as the t-year *spot interest rate*. The investments considered should be zero-coupon bonds, that is, they do not return interest during their term but only at their end.

Only a few time points will have market data or known yields, corresponding to bonds which are available in the bond market, and which are comparable in terms of their origin and risk levels. Yields for other terms are interpolated from the known yields. Different models have been defined for the shape of yield curves. One of the most widely used is the Nelson-Siegal (NS) model for yield curves (Figure 3), which assumes that the yield satisfies a formula

$$y(t) = \beta_1 + \beta_2 * (1 - exp(-t/\lambda_1))/(t/\lambda_1) + \beta_3 * ((1 - exp(-t/\lambda_1))/(t/\lambda_1) - exp(-t/\lambda_1))$$

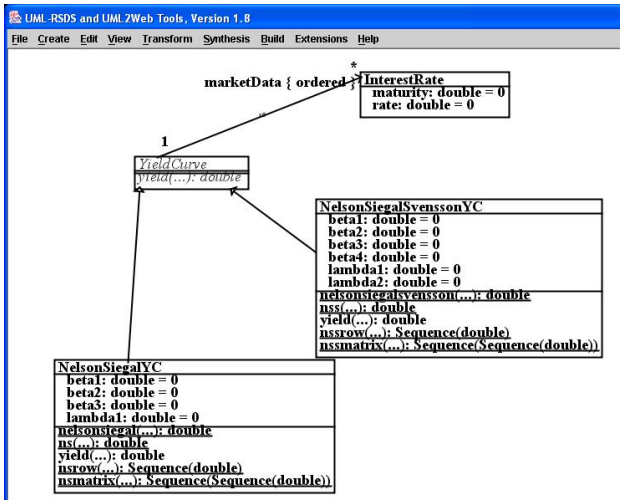

Figure 3. Yield Curve Models.

The yield curve according to the NS model has a long-term rate component (β_1), short-term component (2nd factor), and a 'hump' (3rd factor). The problem is to estimate the β_i and λ_1, for the curve which best fits the given market data: this is termed 'fitting the curve' to the data. Usually some form of sum of squared differences between the estimated curve and the market data is used as a measure of fit. The Nelson-Siegal-Svensson (NSS) model adds a further 'hump' term to the Nelson-Siegal model, with additional parameters β_4 and λ_2. This enables closer fitting of the curve to data in some cases, but also increases the computational cost of the fitting procedure.

The Nelson-Siegal model was specified in UML-RSDS by the following functions of $NelsonSiegalYC$:

```
static query nelsonsiegal(t : double,
       v1 : double, v2 : double,
       v3 : double,
       lambda1 : double) : double
pre: t > 0 & lambda1 > 0
post:
 tscaled1 = t/lambda1 &
 exptscaled1 = (-tscaled1)->exp() &
 expratio1 = (1 - exptscaled1)/tscaled1 &
 result = v1 + v2*expratio1 +
      v3*(expratio1 - exptscaled1)


static query ns(t : double, v1 : double,
         v2 : double, v3 : double,
         lambda1 : double) : double
pre: t >= 0 & lambda1 > 0
post:
 (t = 0 => result = v1 + v2) &
 (t > 0 =>
   result = NelsonSiegalYC.
     nelsonsiegal(t,v1,v2,v3,lambda1))
```

The steps of the yield curve fitting procedure are:

(1) Create *InterestRate* instances for the market data
(2) Produce a number of $NelsonSiegalYC$ instances which provide possible estimated parameters to fit the data, using a commercially-confidential algorithm
(3) Select the best of these, using a sum of squared differences measure of fit, and apply a numerical optimiser to refine the estimates.

For the second stage possible techniques include evolutionary algorithms such as genetic algorithms or differential evolution [28]. We investigated various alternatives and found that a simplified genetic algorithm with a predominance of mutation over crossover was the most effective. This algorithm was also specified in UML-RSDS, and was subsequently reused for a different domain [29].

The NS function is used in the genetic algorithm to evaluate the fitness of individuals, based on the sum of squares of differences between the actual and predicted interest rates for the market data:

```
query fitness() : double
post:
 sumsquares =
  InterestRate->collect( r |
   (r.rate - NelsonSiegalYC.ns(
       r.maturity, traits[1].value,
       traits[2].value, traits[3].value,
       1.7))->sqr() )->sum() &
 result = 100.0/(1 + sumsquares->sqrt())
```

Here, a fixed λ_1 value of 1.7 is used. *traits* are the data items of the GA individual, representing β_1, β_2, β_3.

For the final stage we use a numerical optimisation algorithm based on the simplex algorithm (Figure 4).

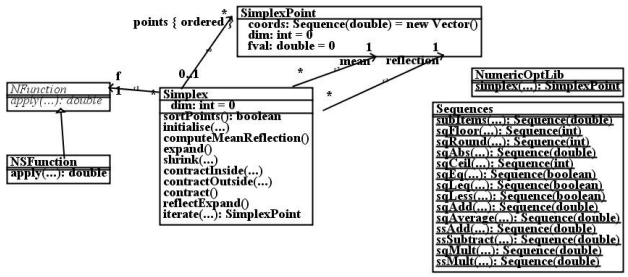

Figure 4. Simplex Class Diagram.

The Nelder-Mead simplex algorithm performs a deterministic search to find a point where a minimal value of the associated n-ary function f is attained, $n > 1$. The attribute *dim* is the dimension n of the search space, the simplex consists of $n+1$ points, each point has n double-valued coordinates.

We formalised the simplex algorithm in UML-RSDS by a series of operations of the class *Simplex*. This component was also recognized as useful for future projects and hence was added to a UML-RSDS finance library.

The key steps of the algorithm are defined by:

```
initialise(x : SimplexPoint)
post:
  x : points &
  Integer.subrange(1,x.dim)->forAll( i |
    SimplexPoint->exists( p |
      p.coords =
        (x.coords.subrange(1,i - 1) ^
         Sequence{ (x.coords->at(i))*1.05 } ^
         x.coords.subrange(i+1,x.dim) ) &
      p.dim = x.dim & p : points ) ) &
  points->forAll( p | p.fval = f.apply(p))

.....

shrink(x1 : SimplexPoint)
post:
  Integer.subrange(2, dim+1)->forAll( i |
    points[i].coords =
      Sequences.ssAdd(x1.coords,
        Sequences.sqMult(
           Sequences.ssSubtract(
             points[i].coords,
             x1.coords), 0.5)) &
    points[i].fval = f.apply(points[i]) )

iterate(tol : double) : SimplexPoint
activity:
  execute sortPoints() ;
  while points[1].fval > tol
  do
    execute computeMeanReflection() &
       reflectExpand() & sortPoints() &
       points[1]->display() ;
  return points[1]
```

The *Simplex* :: *iterate*(*tol* : *double*) operation has an activity to control the execution of the other simplex operations, instead of a postcondition. It is called by the following operation of the *NumericOptLib* library:

```
static simplex(f : NFunction,
  x : Sequence(double),
  tol : double) : SimplexPoint
pre: tol > 0
post:
  SimplexPoint->exists( p |
    p.dim = x.size &
    p.coords = x &
    Simplex->exists( sx |
       sx.f = f & sx.dim = x.size &
       sx.initialise(p) &
       result = sx.iterate(tol) ) )
```

In conclusion, this case study showed that a successful outcome is possible for the XM approach to 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.

5 Lessons Learnt in the Case Studies

Overall, we found that XM could be successfully used in finance case studies. The main challenges were:

- The novelty of application of MDE/MBD in this domain meant that no existing guidelines or examples were available to assist in the specification process.
- The requirements were focussed on functionalities, and a tendency to a pure functional decomposition architecture needed to be counteracted by compositional structuring in terms of appropriate classes based on domain concepts.
- Small modifications were often needed to generated code as part of experimentation and prototyping, instead of exclusive use of executable models. Code changes were however manually reflected back into the specifications for final code generation.

It was found that use cases provided a good structuring mechanism for financial applications. Computation steps within a financial process can be expressed as successive postconditions within a use case, and separate stages in a process can be defined as separate use cases, which are then included in a use case which coordinates the sequencing of the stages. This is in contrast to the data-flow oriented organisation of Excel spreadsheets, which obscures distinct compu-

tational stages.

6 Relationship of XM to Other Methods

Other approaches which aim to integrate agile and MDE are as follows.

MDD-SLAP Zhang and Patel in [14] proposed this method, integrating the System-Level Agile Process (SLAP) and an MDD process in Motorola, with the aim of accelerating the development rate, improving the system quality, and shortening the delivery cycle. SLAP is an agile process that uses Scrum as the baseline and includes some XP practices. Each iteration consists of three phases: (i) application requirements and architecture, (ii) development, (iii) system integration feature testing (SIFT). Motorola's MDD is based on the UML unified software development process (UP), following a V-model process. In order to achieve Agile MDD [14] map activities of MDD into SLAP activities. Although this work is one of the earliest Agile MDD methods, it only describes the general course of development activities without describing the development details such as metamodels and construction of transformations. In contrast, XM specifically mandates minimal (lean) metamodels and transformations. From MDD-SLAP we have adopted the three-phase iteration concept for XM.

Hybrid-MDD Guta et al. [15] propose an MDD process that is integrated with a traditional iterative and incremental process for developing enterprise web applications. The software development process is based on close collaboration between three different teams: the agile development team, the business analyst team, and the MDD team. The process starts with an initial phase which includes the following activities:

- Domain model extraction by the business analyst team and the MDD team.
- Software architecture elaboration by the agile development team.
- MDD environment setup by the MDD team.

Then the process follows an iterative approach whereby the following activities are performed in parallel:

- The business analyst team extends the domain model.
- The agile development team develops hand-coded parts of the system.
- The MDD team develops the different templates and MDD environment.

However the approach does not explicitly incorporate agile development practices. In contrast, in XM we mandate specific agile principles and practices. From Hybrid MDD we adopted the idea of a parallel tooling team providing MDD tool support for the main application development.

RAP Method Basso et al. [30] propose a MDD-based Rapid Application Prototyping (RAP) method. They define four main phases in the development lifecycle:

- Agile analysis: to define the requirements of the system and express it as user stories, sketches and conceptual models.
- Evolutionary prototyping: to generate mock-up models, which are web front ends that provide semantics for web applications.
- Architecture prototyping: to generate annotated UML models from mock-up models developed in evolutionary prototyping.
- Functional prototyping: to generate a working prototype.

The authors introduced RAP into a company which uses Scrum as a development process and organised the process as follows:

- Pre-game: all the planning activities such as product backlog, team selection and release estimation are carried out.
- Game: the proposed RAP method is executed.
- Post-game: the working increment is integrated.

However, this work is designed specifically to support RAP methodology and is primarily applicable to web application development. Moreover, they did not propose an explicit method for integrating agile and MDD.

Agile REMICS Krasteva et al. [31] describe the Agile REMICS method, which supports the modernisation of legacy systems to service cloud by using MDD. They extend their general REMICS migration approach with agile practices based on Scrum. The approach is implemented based on the so-called Type C Scrum in which multiple Scrums overlap. The approach defines five modernisation Scrum types, each of which has its own iterations, product backlog and team. These types are:

- Requirements Scrum: to identify the system requirements.
- Recovery Scrum: to recover the system requirements from the existing code of the legacy system.
- Migration Scrum: to develop the system components.
- Integration and Validation Scrum: to integrate and validate the developed components.
- Control and Supervise Scrum: to monitor the deployed system.

Besides these Scrum types, some XP practices are used such as pair modelling, collective model ownership and continuous modelling. Although this approach promotes the integration of agile development and MDD, it does not describe any MDD related practices such as transformations or metamodelling. Moreover, this approach is specialised to large modernisation developments with multiple teams.

7 Conclusions

We have described an agile MBD process, XM, and illustrated its application on a financial services system. In our experience, UML-RSDS with XM is highly effective for small to medium scale developments. While such a 'model-only' approach may indeed seem 'extreme', in practice it is less time-consuming and less error-prone than a conventional MBD approach working with both code and models. In order to identify the impact of XM approach on the properties of the software development, in future larger case studies with larger development teams will be considered. Additionally, more features such as productivity, time-to-market and comprehensibility of the developed code/specification should also be measured in future studies.

References

[1] K. Beck et. al. Principles behind the agile manifesto. *Agile Alliance*, pages 1–2, 2001.

[2] Version One. 9th annual state of agile survey, 2015.

[3] K. Beck and E. Gamma. *Extreme programming explained: embrace change*. addison-wesley professional, 2004.

[4] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

[5] L. Vijayasarathy and D. Turk. Agile software development: A survey of early adopters. *Journal of Information Technology Management*, 19(2): 1–8, 2008.

[6] F. Grossma, J. Bergin, D. Leip, and O. Gotel. One xp experience: introducing agile (xp) software development into a culture that is willing but not ready. In *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pages 242–254. ACM, 2004. doi:10.1145/1034914.1034933.

[7] M. Church. Why 'agile' and especially scrum are terrible. `https://michaelochurch.wordpress.com`, Date Accessed: June 6, 2015.

[8] J. Greenfield and K. Short. Software factories: Assembling applications with patterns, frameworks, models and tools. *Wiley*, 2004.

[9] R. France and B. Rumpe. Modeling to improve quality or efficiency? an automotive domain perspective. *Software and Systems Modeling*, 11(3): 303, 2012. doi:10.1007/s10270-012-0251-2.

[10] J. Whittle, J. Hutchinson, and Mark. Rouncefield. The state of practice in model-driven engineering. *IEEE software*, 11(4):513–526, 2012. ISSN 1619-1366. doi:10.1007/s10270-012-0261-0.

[11] Bran Selic. What will it take? a view on adoption of model-based methods in practice. *Software & Systems Modeling*, 11(4):513–526, 2012. ISSN 1619-1366. doi:10.1007/s10270-012-0261-0.

[12] S. J.Mellor, M. Balcer, and I. Foreword By-Jacoboson. *Executable UML: A foundation for model-driven architectures*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[13] Jr. James Kirby. Model-driven agile development of reactive multi-agent systems. In *30th Annual International Computer Software and Applications Conference (COMPSAC06)*, pages 297–302. IEEE, 2006. ISBN 0-7695-2655-1. doi:10.1109/COMPSAC.2006.144.

[14] Y. Zhang and S. Patel. Agile model-driven development in practice. *IEEE software*, 28(2):84 – 91, 2011. ISSN 0740-7459. doi:10.1109/MS.2010.85.

[15] J. Novacek, A. Ahari, A. Cornaglia, F. Haxel, A. Viehl, O. Bringmann, and Wo. Rosenstiel. A lightweight mdsd process applied in small projects. In *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, pages 255–258. IEEE, 2009. ISBN 978-0-7695-3784-9. doi:10.1109/SEAA.2009.63.

[16] S. Hansson, Y. Zhao, and H. Burden. How mad are we? empirical evidence for model-driven agile development. In *XM@MoDELS*, 2014.

[17] K. Lano. The uml-rsds manual, 2014.

[18] M. Strittmatter, G. Hinkel, M. Langhammer, R. Jung, and R. Heinrich. Challenges in the evolution of metamodels: Smells and anti-patterns of a historically-grown metamodel. *Workshop on Models and Evolution*, pages 30–39, 2016. ISSN 1613-0073.

[19] C. M. Gerpheide, R. R. H. Schiffelers, and A. Serebrenik. Assessing and improving quality of qvto model transformations. *Software Quality Journal*, 24(3):797–834, 2016. ISSN 0963-9314. doi:10.1007/s11219-015-9280-8.

[20] S. Fang and K. Lano. Extracting correspondences from metamodels using metamodel matching. *PhD symposium, STAF 2019*, 2019.

[21] A. Kästner, M. Gogolla, and B. Selic. From (imperfect) object diagrams to (imperfect) class diagrams. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 13–

22. ACM, 2018. ISBN 978-1-4503-4949-9. doi:10.1145/3239372.3239381.

[22] K. Lano and S. Kolahdouz-Rahimi. Model-transformation design patterns. *IEEE Transactions on Software Engineering*, 40 (12):1224–1259, 2014. ISSN 0098-5589. doi:10.1109/TSE.2014.2354344.

[23] K. Lano and S. Kolahdouz-Rahimi. Constraint-based specification of model transformations. *Journal of Systems and Software*, 86(2):412–436, 2013. doi:10.1016/j.jss.2012.09.006.

[24] OMG. Semantics of a foundational subset for executable uml model (full), 2015.

[25] K. Lano and S. Kolahdouz-Rahimi. Slicing of uml models using model transformations. In *International Conference on Model Driven Engineering Languages and Systems*, pages 228–242. Springer, Berlin, Heidelberg, 2010. ISBN 978-3-642-16128-5. doi:10.1007/978-3-642-16129-2_17.

[26] S. Kolahdouz-Rahimi, K. Lano, S. Pillay, J. Troya, and P. Van Gorp. Evaluation of model transformation approaches for model refactoring. *Science of Computer Programming*, 85:5–40, 2014. doi:10.1016/j.scico.2013.07.013.

[27] K. Lano and S. Yassipour Tehrani. Solving the ttc 2014 movie database case with uml-rsds. In *TTC@ STAF*, pages 150–154, 2014.

[28] M. Gilli, S. Grosse, and E. Schumann. Calibrating the nelson- siegal-svensson model. *SSRN*, 2010. doi:10.2139/ssrn.1676747.

[29] K. Lano, S. Yassipour Tehrani, and S. Kolahdouz Rahimi. Solving the class responsibility assignment case with uml-rsds. In *TTC@ STAF*, pages 9–14, 2016.

[30] F. Basso, R. Pillat, F. Roos-Frantz, and R. Z. Frantz. Combining mde and scrum on the rapid prototyping of web information systems. *International Journal of Web Engineering and Technology*, 10(3):214–244, 2015. doi:10.1504/IJWET.2015.072347.

[31] I. Krasteva, S. Stavros, and S. Ilieva. Agile model-driven modernization to the service cloud. In *The Eighth International Conference on Internet and Web Applications and Services (ICIW 2013). Rome, Italy*, 2013.

Shekoufeh Kolahdouz-Rahimi is an Assistant Professor in the Software Engineering Department at the University of Isfahan. She is an active member of the Model Driven Software Engineering Research Group at this University. She has completed her PhD in Computer Science at Kings College London in 2013. Her research interest includes Design patterns for Model Transformation, Specification and Verification of Model Transformations, Bidirectional Model Transformations, Domain-Specific Modelling Languages and Search-Based Software Engineering.

Kevin Lano is Reader in Software Engineering at King's College London. He has worked for over 25 years in the fields of system specification and verification. A co-founder of the Precise UML group in 1996, he produced some of the first research on model transformation specification and verification and subsequently has developed techniques for the correct-by-construction software engineering of model transformations and for the verification of transformations. He is the author of the UML-RSDS toolset for precise model-based development.

Hessa Alfraihi is a lecturer at the Information Technology department in Princess Nourah bint Abdulrahman University. She completed her PhD and Master degrees in Software Engineering from King's College London and a Bachelor degree in Information Technology from King Saud university. Her research interests include Model-Driven Development (MDD), Agile Software Development (ASD), and empirical research.

Howard P. Haughton has worked in the fields of development and quantitative finance, risk management and has expertise in artificial intelligence and the formal aspects of software engineering. Formally at JP Morgan, Dresdner Bank, Deutsche bank, Merrill Lynch and the Commonwealth Secretariat, he is the director of Holistic Risk Solutions Ltd and Manifest Capital Ltd.