State of Practice in Modeling and Model-Driven Development

Dr. Darius Šilingas
Head of Solutions Department @ No Magic Europe
darius.silingas@nomagic.com
Dr. Darius Šilingas

- Head of Solutions Department @ No Magic Europe
- Head of BPM studies @ ISM Executive School
- Expert in information system and business modeling, lead 200+ training/consulting sessions in 22 countries
- Chair of an annual conference “Business Process Management in Practice” in Lithuania
About MagicDraw

- A popular UML-based modeling platform
- Available since 1998
- Over 500,000 installations in 90+ countries
- Standard-compliant
- Applies MDA for R&D activities
- Designed for customization to customer needs

More info: www.magicdraw.com

Awards

- Jolt Productivity Winner
- Best Java Database Tool
- Jolt Productivity Winner
- Best Java Modeling Tool
- Best Team Development Tool
Meet No Magic

4000+ trainings

1,000,000+ installations

10,000 companies

90 in countries

Allen, TX, USA

Kaunas, Lithuania

Bangkok, Thailand
No Magic Solutions for Enterprises

- Systems engineering
- Software engineering
- Enterprise architecture
- Business process modeling
Software Engineering: a Classical View

What the user asked for

How the analyst saw it

How the system was designed

As the programmer wrote it

What the user really wanted

How it actually works
There is no single development, in either technology or management technique, which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity


Is Model-Driven Development a Silver Bullet?
Modeling Solution is a combination of modeling language(s), methodology and tool(s) that provide a productive infrastructure for applying model-driven development in context of a particular organization.
Evolution of Modeling Languages and Methods

- **1995**: Unified Modeling Language (UML) ← Booch, OMT, OOSE
- **1997**: UML 1.1 is adopted by Object Management Group (OMG)
- **1998**: Release of MagicDraw 1.0
- **2001**: Model-Driven Architecture (MDA)
- **2005**: UML 2.0 – a major revision that replaced UML 1.5
- **2006**: Systems Modeling Language (SysML) 1.0
- **2007**: Model Based Systems Engineering (MBSE) Initiative by INCOSE
- **2011**: Business Process Model and Notation (BPMN) 2.0

Currently:

- UML 2.5, SysML 1.4, BPMN 2.0, fUML, OSLC, UAF, Internet of Things, Business Architecture, professional certifications (OCUP, OCEB, OCSMP)
Using UML in Software Workflows

- Domain concepts and relations
  - Domain object lifecycle
  - Business processes
  - Actors and use cases
  - Use cases scenarios

- Package/component structure
  - Interaction scenarios
  - Data structure
  - Service API
  - GUI navigation schemas

- Code generation from UML
- Visualization of code structure
- Model transformations

- Test case action flows
  - Test data object structures
  - Interactions for test scenarios

- Requirements

- Design

- Testing

- Programming
Applying UML in Software Development Process

- Domain concepts
  - Domain object lifecycle
  - Business processes
  - Actors and use cases
  - Use cases scenarios

- Package/component structure
  - Interaction scenarios
  - Data structure
  - Service API
  - GUI navigation & prototypes

Requirements → Design → Testing → Programming

- Test case action flows
- Test data object structures
- Interactions for test scenarios

- Code generation from UML
- Visualization of code structure
- Model transformations
Conceptual modeling focuses on defining terms and their relations, not so much on precise data properties.
Domain Concept Lifecycle

- **Draft**
  - Confirm
  - Edit

- **Ready**
  - at (activePeriod.start)

- **Active**
  - ExtendActivePeriod
  - at (activePeriod.end)

- **Closed**
Business Process Model (BPMN)

1. Class Start
2. Give Lecture
3. Lecture Time
4. Mid-Term Time
5. Take Mid-Term Test
   - Mid-term Test: Test
   - Mid-term Assessments: Assessment
6. Final Exam Time
7. Take Final Exam
   - Final Exam Test: Test
   - Final Exam Assessments: Assessment
8. Finalize Student Grades
Use Cases

Student
- Perform Test Assessment

Teacher
- Compose Test
- Define Question
- Discard Question
Use Case Scenarios – Perform Test Assessment

<table>
<thead>
<tr>
<th>Student</th>
<th>MagicTest Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Test</td>
<td>Display Instructions</td>
</tr>
<tr>
<td>Start Test Assessment</td>
<td>Display Question</td>
</tr>
<tr>
<td>Provide Answer</td>
<td>Register Answer</td>
</tr>
<tr>
<td>Review Score</td>
<td>Calculate Score</td>
</tr>
</tbody>
</table>

[more questions]

[no more questions]
Applying UML in Software Development Process

- Domain concepts
- Domain object lifecycle
- Business processes
- Actors and use cases
- Use cases scenarios

- Package/component structure
- Interaction scenarios
- Data structure
- Service API
- GUI navigation & prototypes

- Requirements
- Design

- Testing
- Programming

- Test case action flows
- Test data object structures
- Interactions for test scenarios

- Code generation from UML
- Visualization of code structure
- Model transformations
System context diagram is often included in project vision in order to understand IT solution integration environment.
Component Dependencies

«application»
MagicTest

«service»
TestComposer

«service»
TestAssessment

«service»
ClassManager

«service»
PersonnelManager

«database»
University Data Warehouse
Package Dependencies

MagicTest
- Answer
- Test
- Assessment
- SelectionAnswer
- InputAnswer
- Question
- AnswerOption
- OpenQuestion
- ClosedQuestion
- TestStatus
- Period
- MagicTest Data Structure

MagicUniversity
- Teacher
- Class
- Course
- Student
- UniversityMember
- AcademicRole
- MagicUniversity Data Structure
package MagicTest

Test
- author : Teacher [1]
- title : String [1]
- instructions : String [1]
- allowedTime : Integer [1]
- targetClass : Class [1]

Question
- description : String [1]
- subjects : Course [1..*]
- author : Teacher [1]

Answer
- correct : boolean [1]
- registered : date [1]

TestStatus
- DRAFT
- COMPLETE
- DEPRECATED

Assessment
- author : Student [1]
- started : date [1]
- ended : date [1]
- grade : Integer [1]

OpenQuestion
- expectedAnswer : String [1]

ClosedQuestion
- options : String [2..*]

SelectionAnswer
- input : String [1]

AnswerOption
- statement : String [1]
- correct : boolean [1]
UI Screen Prototype: Test List

Test List

<table>
<thead>
<tr>
<th>UML Basics</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business English Mid-Term</td>
<td>Trial test to check student's knowledge about UML before the mid-term test. The score will not be included into course grade calculation.</td>
</tr>
<tr>
<td>Statistics Mid-Term</td>
<td></td>
</tr>
</tbody>
</table>

Questions: 2

Allowed time: 5 min.

Start Assessment  Close
Question: 1/2

Remaining time: 5 min.

How many metaclasses are defined in UML 2

Enter Your Answer

Back  Next  Finish
Question: 2/2
Remaining time: 3 min.

How many diagrams are defined in UML 2

- 9 diagrams
- 15 diagrams

Back  Next  Finish
UI Navigation Schema: Test Assessment Results

Test List

Select Test

Test Assessment

OK

End Assessment

Test Assessment Results

Login

Quit
Test: UML Basics
Student: Darius Silingas
Questions: 2
Answers: 2
Correct Answers: 2
Incorrect Answers: 0

Comment: Great! No weak areas.

Score: 10
Applying UML in Software Development Process

- Domain concepts
- Domain object lifecycle
- Business processes
- Actors and use cases
- Use cases scenarios

- Package/component structure
- Interaction scenarios
- Data structure
- Service API
- GUI navigation & prototypes

- Requirements
- Design
- Testing
- Programming

- Test case action flows
- Test data object structures
- Interactions for test scenarios

- Code generation from UML
  Visualization of code structure
  Model transformations
Model Transformations

![Diagram of model transformations]

**TEMPLATE**

This simple template will list all classes from selected packages. The context of each class consists of class name, visibility, and its operations.

```
\$Class
\$class.name

\#foreach (\$cls in \$Class)
\$cls.name
\#end
```
<xs:schema targetNamespace="MagicTest">
    <xs:complexType name="Answer"></xs:complexType>
    <xs:complexType name="AnswerOption"></xs:complexType>
    <xs:complexType name="Assessment"></xs:complexType>
    <xs:complexType name="ClosedQuestion"></xs:complexType>
    <xs:complexType name="InputAnswer"></xs:complexType>
    <xs:complexType name="OpenQuestion"></xs:complexType>
    <xs:complexType name="Question"></xs:complexType>
    <xs:complexType name="SelectionAqnwer"></xs:complexType>
    <xs:complexType name="Test">
        <xs:annotation>
            <xs:documentation>
                Represents a set up of a question set and assessment related properties such as instructions and allowed time.
                A test is always authored by one teacher and is assigned to one particular class. A test is reused for multiple
                assessments authored by the students who attend the class to which the test assigned.
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element maxOccurs="1" minOccurs="1" name="author" type="Teacher"/>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="questions" type="Question"/>
            <xs:element maxOccurs="1" minOccurs="1" name="title" type="xs:string"/>
            <xs:element maxOccurs="1" minOccurs="1" name="instructions" type="xs:string"/>
            <xs:element maxOccurs="1" minOccurs="1" name="allowedTime" type="xs:integer"/>
            <xs:element maxOccurs="1" minOccurs="1" name="state" type="TestState"/>
            <xs:element maxOccurs="1" minOccurs="1" name="target" type="Class"/>
        </xs:sequence>
    </xs:complexType>
    <xs:simpleType name="TestState"></xs:simpleType>
</xs:schema>
Applying UML in Software Development Process

- Domain concepts
  - Domain object lifecycle
  - Business processes
  - Actors and use cases
  - Use cases scenarios

- Package/component structure
  - Interaction scenarios
  - Data structure
  - Service API
  - GUI navigation & prototypes

- Requirements
  - Domain concepts
    - Domain object lifecycle
    - Business processes
    - Actors and use cases
    - Use cases scenarios
  - Test case action flows
  - Test data object structures
  - Interactions for test scenarios

- Design
  - Package/component structure
    - Interaction scenarios
    - Data structure
    - Service API
    - GUI navigation & prototypes
  - Code generation from UML
  - Visualization of code structure
  - Model transformations

- Testing

- Programming
Model Driven Development and Software Development Process

Project Management
- Requirements
- Model-Driven Development

Quality Management
- Design
- Programming

Technologies
- Testing

Development Tools
• A man flying in a hot air balloon suddenly realizes he’s lost.
• He reduces height and spots a man down below. He lowers the balloon further and shouts to get directions, "Excuse me, can you tell me where I am?"
• The man below says: "Yes. You're in a hot air balloon, hovering 30 feet above this field."
• "You must work in Information Technology," says the balloonist.
• "I do" replies the man. "How did you know?"
• "Well," says the balloonist, "everything you have told me is technically correct, but It's of no use to anyone."
• The man below replies, "You must work in management."
• "I do," replies the balloonist, "But how'd you know?"
• "Well", says the man, "you don’t know where you are or where you’re going, but you expect me to be able to help. You’re in the same position you were before we met, but now it’s my fault."
Code is not enough ...
Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it.

We value:

• Individuals and interactions over processes and tools

• Working software over comprehensive documentation

• Customer collaboration over contract negotiation

• Responding to change over following a plan
Critique from Agile Practitioners

• Models are bad, because models:
  - Don’t run
  - Don’t crash
  - Can’t be tested automatically

• Models are bad because they are “documentation”, which:
  - Has no correspondence to the code
  - Is extra work to build…
  - …and maintain (or throw away)
WHY do you model?
Generic Modeling Benefits

1. Understanding complex structures and behaviors
2. Improving communication and collaboration
3. Easier analysis, estimation, experimentation, and improvement planning
4. Knowledge preservation and reuse
Throwaway Modeling

Models are created for a short-term usage
  - Typically for scoping change in IT projects
A particular aspect is emphasized
  - Automation, data exchange, task durations, waste, etc.

Consistency and completeness is not the main concern

✓ Apply simplest tools
✓ Do not forget to throw away the model!
Sustainable Modeling

- Models are corporate knowledge asset that provides a long-term value and needs to evolve

- Strict adherence to the principles is necessary
- A need for a dedicated business architecture tool
- A need for a dedicated team supporting this effort
Realizing Value of Business Architecture

- Center of Excellence
- Principles
- Modeling
- Governing Models
- Using Models
- High Value from Models
Model-Driven Development Scenarios beyond Code Generation

1. Test Driven Modeling

2. Model Driven Requirements Management

3. Architecture Planning & Code Review Against Architecture

4. Model Driven System Documentation
From Model Driven Testing to Test Driven Modeling

- Improve MDA with Agile practices
- Raise the level of abstraction for Agile practices
• Let’s practice example/test-driven approach and learn the principles of Test Driven Modeling by analyzing how to apply it to a small example system MagicTest.

• MagicTest provides functionality for a teacher to create test questions, group them into a test and assign it to the class that he is teaching. The students of that classes make assessments of the test that are automatically evaluated.
How do I know if this data structure is suitable?
A sample shows inconsistency with data structure.
What Can We Do When Sample Violates Data Structure

1. Fix data structure (sample is good, structure was wrong)

2. Fix data sample (sample is wrong, structure was good)

- Use invalid sample in testing for ensuring that such cases are not allowed and handled appropriately in the system
Decision: Enable multiple supervisors
A sample is now in sync with data structure
A data designer should prepare it along with data structure

Testers should take over with creating variations
Modeling Interaction Scenario at Business Logic Layer

Service operations are discovered based on scenarios.
GUI Modeling in UML

GUI structure can be captured in UML class or composite structure diagrams

- Modern modeling tools typically provide graphical rendering to enable better presentation for users and GUI designers/analysts

1. Identify abstract GUI screens as UML classes

2. Build abstract GUI navigation schema as UML state machine

3. Build a story board GUI sample set as UML instance specifications for validating GUI design with data samples
   ✓ Reuse data for GUI samples from data design (or vice versa)
How do I know it is suitable?
Validates both navigation schema and screen structures
Validates both navigation schema and screen structures
Validates both navigation schema and screen structures
Lessons Learned

1. Conceptual mistakes in design models can be found early by validating it with example/test models

2. Quality engineers need to be involved in collaborative modeling together with developers
   - Developers should produce essential example models
   - Testers should take over with creating variations for a better coverage of error-prone situations

3. Example/test-driven modeling enables achieving a better quality of model-driven software by testing design models early
Model-Driven Development Scenarios beyond Code Generation

1. Test Driven Modeling

2. Model Driven Requirements Management

3. Architecture Planning & Code Review Against Architecture

4. Model Driven System Documentation
In 2009, MagicDraw R&D decided to migrate from document-driven to model-driven requirement engineering using SysML.

Advantages:

- Much better teamwork and version management capabilities
- More formal/structured descriptions of the requirements
- Maintain the information about already implemented functionality
- Traceability to the architecture and test cases
- Eating your own dog food

After 6 years of applying model-based requirements engineering, MagicDraw R&D team doesn’t want to go back to documents 😊
• SysML is a specialized UML profile targeted to system engineering (as opposed to software engineering)
• SysML defines elements for modeling requirements and their relationships (including relationships to other artifacts such as test case or block)
Element Layout in Requirement Diagrams
<table>
<thead>
<tr>
<th>#</th>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TWCAT150.5.3.2.2</td>
<td>Buttons</td>
<td>There shall be buttons OK and Cancel. The OK button shall confirm changes.</td>
</tr>
<tr>
<td>2</td>
<td>TWCAT150.5.3.3</td>
<td>Category Management Window</td>
<td>Sketch of the window is attached to this requirement. Also there shall be an input box called &quot;Category name&quot; for entering category names.</td>
</tr>
<tr>
<td>3</td>
<td>TWCAT150.5.2.7.1</td>
<td>Plain Project List</td>
<td>After pressing the Alphabetical View button in the Category Tool, the project list shall be ordered alphabetically.</td>
</tr>
<tr>
<td>4</td>
<td>TWCAT150.5.5</td>
<td>Adding Local Projects to Teamwork</td>
<td>The &quot;Add Project to Teamwork Settings&quot; window (detail view) shall be displayed.</td>
</tr>
<tr>
<td>5</td>
<td>TWCAT150.5.3.2</td>
<td>Category Renaming</td>
<td>Sketch of the window is attached to this requirement. Also there shall be an input box called &quot;Category name&quot; for entering category names.</td>
</tr>
<tr>
<td>6</td>
<td>TWCAT150.5.2.4.2.2</td>
<td>Alphabetical View</td>
<td>This button shall be depressed by default. It allows viewing projects alphabetically.</td>
</tr>
<tr>
<td>7</td>
<td>TWCAT150.5.2.4.2.4</td>
<td>Collapse</td>
<td>This button shall collapse all categories by allowing the user to view the project list more easily.</td>
</tr>
<tr>
<td>8</td>
<td>TWCAT150.9</td>
<td>Category Permissions</td>
<td>There shall be an input box called &quot;Category name&quot; for entering category names.</td>
</tr>
</tbody>
</table>
| 9  | TWCAT150.5.3.3.2    | Window Contents                                           | After pressing the Add button, the new category shall be added. If a category with the same name exists, an error message shall be displayed: "A category with this name already exists."
After showing this error, the category shall not be added to the project list. |
| 10 | TWCAT150.5.2.4.2    | Buttons                                                   | The toolbar shall have 8 buttons. Button names shall be: OK, Cancel, Add, Remove, Alphabetical View, Collapsible, Category Permissions, and Category Rename.                                                     |
| 11 | TWCAT150.5.2.8      | Project List Component Usage in MagicDraw                | Project list component is used in the "Open Teamwork Project", and Project list component uses this component with the Branch column hidden.                                                                 |
Modeling GUI Requirements
Lessons Learned

1. Model-based requirements are much easier to manage and evolve

2. Modeling facilitates defining more structured and better quality requirements

3. Model-based approach enables better collaboration between analysts who define requirements and quality engineers who define test cases verifying them

4. Change is not easy – the idea was floating around for some time but it was implemented only by a new product manager, who invested a lot of effort to make it happen
1. Test Driven Modeling

2. Model Driven Requirements Management

3. Architecture Planning & Code Review Against Architecture

4. Model Driven System Documentation
What Is the Most Popular Architecture Planning Tool?
Model-Code Engineering

Code Generation
- Generate code from model files based on an existing template

Reverse
- Retrieve system architecture from code and visualize it
- **Note #1**: in complex systems developers need to fill in metafiles
- **Note #2**: automated visualization is not easy
Lessons Learned

1. While code can reveal “as is” dependencies of implementation components, it is not useful as a planning tool.

2. Visual modeling with UML is a good means for architecture planning.

3. UML needs to be extended with a customer-specific profile, which captures important context information such as interface levels, responsible teams, technologies, etc.

4. It is necessary to map code to architecture in order to perform code-architecture reviews.
1. Test Driven Modeling

2. Model Driven Requirements Management

3. Architecture Planning & Code Review Against Architecture

4. Model Driven System Documentation
What is the Most Popular System Documentation Tool?
Generating Documents from Model

You can generate an HTML, Reach Text and Open Office documents, XML or any other simple text report for a modeling project.
Report as a Model Transformation

- **Model 2 Model**
- **Model 2 Code**
- **Model 2 Document**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader</td>
<td>Information about library customer.</td>
</tr>
<tr>
<td>Request</td>
<td>Document registering reader's wish to have a new title in a library.</td>
</tr>
<tr>
<td>Request Evaluation</td>
<td>Librarian's decision whether to approve or deny reader's request.</td>
</tr>
<tr>
<td>Title</td>
<td>Information about a book, journal or another kind of library inventory item. Library may contain multiple copies of the same title.</td>
</tr>
</tbody>
</table>
Lessons Learned

1. Models may be a good choice as master source for system documentation

2. Model-driven system documentation facilitates a more structured system documentation, a separation of style and content, and elimination of duplicated information in a master source

3. A good rule of thumb telling if it is a good idea to go for a model-driven system documentation is if a major/bigger part of system documentation content is depicted in figures, tables, and lists.
Consider code generation as an integral part of a modeling solution, focus on model(ing) value in your organization’s context.
Thank You for the attention!

Any Questions

Let’s Keep in Touch:

e-mail: darius.silingas@nomagic.com
Skype: darius.silingas
Get connected at LinkedIn social network