uOttawa

L'Université canadienne
Canada's university

Panel: Obstacles to MDE Adoption

CRuiSE: Complexity Reduction in Software Engineering

May 2008

Professor: Dr. Timothy C. Lethbridge
Students: Andrew Forward, Dusan Brestovansky, Omar Bahy
IBM Ottawa Collaborator: Marcellus Mindel

University of Ottawa

Some key objectives of the CRuiSE group:

Better understand the roots of software system complexity

- From the perspective of all stakeholders
  - Software engineers, users, configurers, etc.
- What actions and activities drive complexity?

Better understand the successes and failures of existing tools, when it comes to complexity reduction

- E.g. Modelling in general, UML, BPEL

Levels of MDE

Model-only: Approaches where the model is effectively all there is, except for small amounts of code.

Model-centric: Approaches where modeling is performed first, and code is generated from the model, for possible subsequent manual manipulation.

Model-as-design-aid: Approaches where modeling is done for design purposes, but then code is written mostly by hand.

Model-as-documentation: Approaches where modeling is done to outline or describe the system, largely after the code is written.

Code-only: Approaches where modeling is almost entirely absent.

Goals of MDE

- **Balancing between creating good software now, with enabling future development of good software**

In particular to MDE

- **Enable as much development at modelling level**
  - Without need to edit generated code
- **Enable more rigorous *engineering* at modelling level**
  - analysis, testing, verification

Achieved relatively soon, but late adopters might take 20 or more years

Current State of the Art

- **A lot of emphasis on artifacts that are great at communicating design and intent**
  - Must also consider how design is *written*
  - Tools lacking efficient process to create, edit and maintain software models
- **Tools are big and clunky**
  - We need big tools for big projects
  - We ALSO need small tools for small projects

Benefits / Drawbacks to MDE

Main Benefit

- **Documentation / Communication**
  - Visually document your design / intent for *free*
  - System is built visually from the beginning

Main Drawback

- **Training**
  - Not only is modelling a skill, but the tools require significant investment of training

Other Drawbacks

- **Inefficient *programming*, tool specific dependency, adoption, difficulty (still) capturing business logic**

Why (or why not) MDE

Biggest Obstacle

- **Must be efficient for software developers to build software systems (create, edit, maintain, debug)**

Best Reason to MDE

- **Allows higher order constructs (i.e. more abstract 1st class constructs)**

Best Reason NOT to

- **Tool support (limited, vendor specific, interoperability issues)**

Empirical Study: Attitudes to Modelling

Objective: Understand how software engineers think about modelling, and do modelling so we can develop better tools

Asking various questions such as:

- What is a model?
- How do you create / modify software models
- How do you learn about the design of software
- What modelling notations do you use?

References for more information

Andrew Forward's PhD Homepage

http://www.site.uottawa.ca/~tcl/gradtheses/aforwardphd/

Computer Science PhD Thesis by Andrew Forward

Text-Diagram Duality: How Code-Centric and Model-Centric Languages, Models, and Programming can Co-Exist

Links to important components of the thesis:

- Survey on Software Modeling - Questions The questions asked on a survey of software practitioners about software modeling. Conducted April 2007 to December 2007.
- Survey On Software Modeling - Technical Report [last updated: Feb 8, 2007]. Work in progress. A technical summary of the of results of the the survey presented above.
- Survey On Software Modeling - Sub Samples [last updated: Feb 8, 2007]. Work in progress. A companion document to the technical report that provides additional sub sampling.
- Software Application Taxonomy. We used a systematic process to develop a software application taxonomy to cover all types of software currently being developed. The taxonomy should be useful to both researchers and practitioners who need to perform such tasks as cataloguing, filing or searching for applications. The taxonomy will also facilitate tagging of components and techniques according to the application types they are suitable for. Conducted March 2007 to June 2007.

Empirical Study: Overview

- April – December 2007
- 113 Participants
- 18 questions

Demographics

- 14 years of SE experience
- 2/3 from Canada and United States
- Other regions include United Kingdom, Europe, India, Pakistan, Australia, Mexico and Singapore
- 44% had a Masters degree

Key Findings: Creating Vs Consuming Models

uOttawa
L'Université canadienne
Canada's university

Creating Models

- White board, diagramming tools, word-processors

Consuming Models

- Word-of-mouth, word-processors, diagramming tools

Special Notes

- Least important source of information: fully integrated modeling tools and hand-written material

Key Findings: Use of modelling tools

Top Uses

- **Develop the design of software**
- **Transcribe a design into a digital format**

Bottom Uses

- **Code generation**
- **Brainstorming design alternatives (whiteboards still superior?)**

Key Finding: Code versus Model-Centric

Responses for Question 14: Tasks that are better in a model-centric versus code-centric approach.

Available activities	% Much easier in Models (1)	% Easier in Models (1 + 2)	% Easier in Code (4 + 5)	% Much easier in Code (5)
Fixing a bug	21.1	28.9	43.3	25.6
Creating efficient software	16.3	35.9	43.5	21.7
Creating a system as quickly as possible	23.9	46.7	42.4	23.9
Creating a prototype	26.7	43.0	32.6	22.8
Creating a usable system for end users	26.1	42.4	22.8	10.9
Modifying a system when requirements change	34.1	54.9	24.2	13.2
Creating a system that most accurately meets requirements	42.9	67.0	19.8	8.8
Creating a re-usable system	44.6	63.0	15.2	9.8
Creating a new system overall	43.5	68.5	20.7	7.6
Comprehending a system's behavior	51.7	71.9	15.7	5.6
Explaining a system to others	61.1	81.8	7.6	6.5

Note. Values range from Much easier in a model-centric approach (1), to much easier in a code-centric approach (5).

Key Findings: Problems with Model-Centric

uOttawa
L'Université canadienne
Canada's university

Responses for Question 15: Problems with a model-centric approach.

Potential problems	% Strongly Disagree (1)	% Disagree (1 + 2)	% Agree (4 + 5)	% Strongly Agree (5)
Models become out of date and inconsistent with code	7.6	16.3	68.5	37.0
Models cannot be easily exchanged between tools	15.4	26.4	51.6	17.6
Modeling tools are 'heavyweight' (to install, learn, configure, use)	10.9	31.5	39.1	12.0
Code generated from a modeling tool not of the kind I would like	18.7	39.6	38.5	16.5
You cannot describe the kinds of details that need to be implemented	23.6	43.8	36.0	7.9
Creating and editing a model is slow	17.4	43.5	22.8	12.0
Modeling tools change, models become obsolete	22.8	44.6	32.6	5.4
Modeling tools lack features I need or want	19.1	44.9	21.3	5.6
Modeling tools hide too many details that are visible in the source code	19.6	44.6	23.9	1.1
Modeling tools are too expensive	26.7	46.7	26.7	6.7
Modeling tools do not allow be to analyze my design in ways I would want	28.9	51.1	25.6	6.7
Organization culture does not like modeling	31.5	48.9	23.9	4.3
Semantics of models different from underling prog. languages	31.1	56.7	23.3	8.9
Modeling languages are not expressive enough	28.6	54.9	17.6	2.2
Modeling language hard to understand	28.6	62.6	9.9	3.3
Have had bad experiences with modeling	39.6	63.7	16.5	6.6
Do not trust companies will continue to support their tools	44.9	67.4	10.1	0.0

Note. Values range from Not a problem (1), to Terrible problem (5).

Key Findings: Problems with Code-Centric

Responses for Question 16: Problems with a code-centric approach.				
Potential problems	% Strongly Disagree (1)	% Disagree (1 + 2)	% Agree (4 + 5)	% Strongly. Agree (5)
Hard to see overall design	4.3	13.8	66.0	35.1
Hard to understand behavior of system	4.3	19.1	60.6	21.3
Code becomes of poorer quality over time	9.8	28.3	55.4	25.0
Too difficult to restructure system when needed	8.6	22.6	51.6	17.2
Difficult to change code without adding bugs	9.7	22.6	50.5	18.3
Changing code takes too much time	20.2	39.4	27.7	8.5
Our programming language leads to complex code	26.6	51.1	20.2	8.5
More skill is required than is available to develop high quality code	29.7	53.8	22.0	6.6
Programming languages are not expressive enough	46.2	64.8	14.3	5.5
Organization culture does not like the code-centric approach	58.7	72.8	14.1	4.3
Our programming language is likely to become obsolete	51.6	75.3	9.7	3.2

Note. Values range from Not a problem (1) to Terrible problem (5).