The Pragmatics of Model-Driven Development

Bran Selic, IBM Rational Software
Outline

- Define MDD
- Code generation (rant)
- Quality
- Pragmatics
- Conclusion + comments
Introduction

- Software is evolving slowly
- 3rd generation languages began in late 1950’s
- Object Oriented Design
- “Even Classes and Inheritance are underused”
What is MDD?

- Using models as primary artifact
- Automation
- Model transformation
- Model verification
- Code generation
- NOT a set of rules to follow
- It is an IDEA – Where is the line?
Acronyms

- MDA
- MDE
- MDD
- MDSD
- MDSE

- MDE first coined by Stuart Kent (U of Kent, 2001).
Why MDD?

- Models are easier to:
  - Specify
  - Maintain
  - Understand
Why do we need it

- MDE aims to:
  - Improve short term productivity
  - Improve long term productivity
  - Improve software ROI
  - Reduce sensitivity to change – artifacts are used longer
Why do we need it

- Faster
- Less prone to errors
- Automated documentation
- Empowers DOMAIN EXPERTS
Code generation
Code Generation

- Just skeletal?
- Relation of code to model
- Round trip engineering
- DSL code generation
“A key premise behind MDD is that programs are automatically generated from their corresponding models”
There is rarely a need to examine or modify the program directly.
Settings
Language
Set time & date
GPS system
Entertainment
Reset

Time  Temp.  Cons.
And now back to the paper...
Quality of a Model

- Abstraction
- Understandability
- Accuracy
- Predictiveness
- Inexpensive
Executability

- Must have ability to:
  - Start
  - Stop
  - Resume
  - Automated instrumentation

- Executing + experimentation models helps learning
- “Hello world” raises confidence
Executability
Scalability

- Intended for large corporations / large software
- Metrics:
  - compilation time – full and partial
  - Code generation time
  - System size
Efficiency

- Modern compilers outperform manual coding
- Two categories:
  - performance
  - memory management
- Model to code is within 5%-15% efficiency of manual
- Will improve!
- Occasionally need to add manual code to generated code
- Used as an excuse to reject MDD
Integration

- Need compatibility with existing
  - Compilers
  - Build utilities
  - Debuggers
  - Code analyzers
  - Software versioning control systems
  - Legacy code
Integration

- Begin with small projects or extensions
- CANNOT require purchasing of new systems
- Customizable code generators
- Allow direct calls to such utilities from within the model
Why hasn’t it caught on?
CASE
CASE Dejavu

- **Uses:**
  - Diagrammatic Tools
  - Information Repository
  - Interface Generators
  - Management tools
  - Process modelling
  - Others

- **Promised:**
  - high-quality,
  - defect-free
  - maintainable software products
CASE

- Enforced a way of
  - Working
  - Modelling
  - Coding
- Tools didn’t match existing coding practices
- Single tool – not tuned to developer needs
- Resulting code required too much modification
- If code needed editing – must manage TWO systems
- Will MDE encounter same pitfalls?
Now back to MDE

- Fears
  - Generated code not as efficient
  - Wastes time and money
- Focused too much on academics
- Large culture change
- How do we push it to industry?
Why hasn’t it caught on?

- Requires new skill set
- Models are abandoned when too much modification is required.
- High risk!
- Universities teach compilers and parsers. What about DSL, or code generators.
Why hasn’t it caught on?

- Lack of tools, or undeveloped (i.e. version control)
- Using MDD with little experience
- Too many promises (platform independence)
- Learning curve
- Cost of creating the tool
- Are we simplifying the process but making it harder at the same time?
Why hasn’t it caught on

I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared with the conceptual errors in most systems. If this is true, building software will always be hard. **There is inherently no silver bullet.**
Why hasn’t it caught on

For almost 40 years, people have been anticipating and writing about "automatic programming," or the generation of a program for solving a problem from a statement of the problem specifications. Some today write as if they expect this technology to provide the next breakthrough.

“automatic programming always has been a euphemism for programming with a higher-level language than was presently available to the programmer”
Quick Demos

- Visual Studio / WPF
- Labview - http://www.youtube.com/watch?v=xpsQFvc7v5E&feature=related
Comments

- Introductory paper
- Not much new
- Who is the target audience?