Are Domain-Specific Models Easier to Maintain Than UML Models?

Lan Cao, *Old Dominion University*
Balasubramaniam Ramesh, *Georgia State University*
Matti Rossi, *Helsinki School of Economics*

Experimental results show that maintenance can be significantly easier and faster with a DSM language than with a general-purpose modeling language.

Although domain-specific modeling (DSM) languages have been adopted in industries such as telecommunications and insurance, they haven't yet gained wide acceptance in practice. This is because the claims of increased productivity and ease of understanding haven't yet been verified by independent studies. To address this concern, we examined a DSM language's performance for maintenance tasks. Maintenance in software-intensive systems is critical because software often continuously evolves during development as well as after delivery, to meet users' ever-changing needs. So, maintenance performance significantly impacts software development productivity.

Proponents claim that a key driver of DSM is easier comprehension of system structure and behavior, which should make evaluating and maintaining the models easier. We investigate this through the following research question: Does DSM improve the maintenance performance of designers, compared to general-purpose modeling using UML?

Our Objectives

Maintenance effort has two components:

- understanding the artifact being changed and the changes' impact on the artifact, and
- incorporating changes.

We investigated how each type of modeling language affects model comprehension, the correctness of changes, and the degree of changes made during a maintenance task.

We assessed the accuracy with which designers understand model syntax and model semantics. Model syntax defines a language's or a representation's forms and structure. DSM directly represents the problem space by mapping modeling concepts to domain concepts. The modeling language incorporates the business rules representing domain knowledge. Furthermore, these domain concepts represent the system's design by specifying the system's static structure and dynamic behavior. As a result, models created with DSM match well with domain specialists' vocabularies.[1] So, we expect designers to comprehend the semantics of DSM models more accurately than that of UML models.

0740-7459/09/$25.00 © 2009 IEEE

Table 1

A comparison of UML and domain-specific modeling in maintenance performance

Dependent variable	Unit	UML	DSM	p value
Syntactic accuracy	The percentage of correct answers	66.4	70.3	0.03
Semantic accuracy	The percentage of correct answers	68.8	76.4	0.03
Correctness of change	The score on a 100-point scale for the changes' correctness	68.5	83.2	<0.01
Degree of change	The number of "steps" involved in incorporating the change, weighted by each step's size	8.7	4.6	<0.01

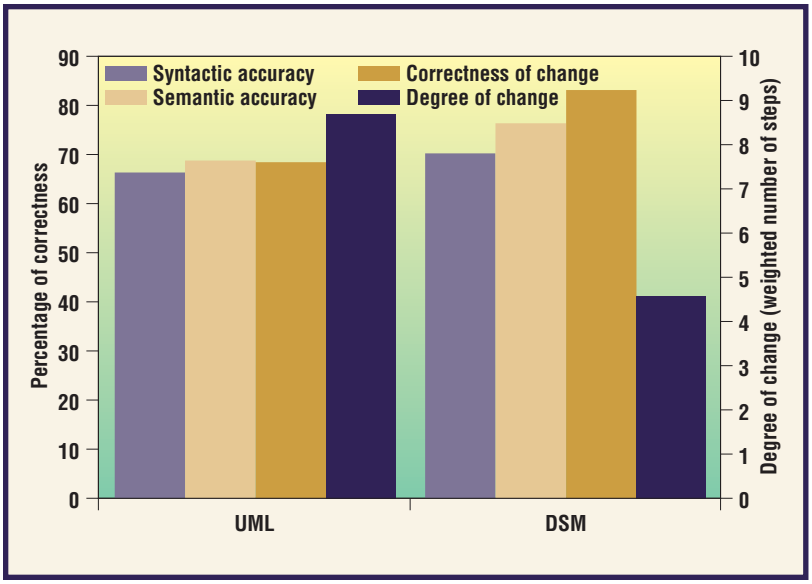


Figure 1. A comparison of comprehension and changeability (the ease of modifying a model) between UML and domain-specific modeling (DSM). DSM is better in both model comprehension and model changeability.

The tight coupling of the DSM with the domain allows a simpler, more compact language. Also, the DSM language can incorporate many business rules and domain concepts, leading to smaller models. In contrast, the tendency to generically use UML often results in large, complex models. So, we expect designers to comprehend the syntax of DSM models more accurately than that of UML models.

The information obtained during design, in the early stages, is usually informal and subject to rapid changes. Therefore, models constructed at this point often undergo major changes. The model's changeability (the ease of modifying it) significantly influences the development project's productivity. A DSM language's higher abstraction levels and smaller size should decrease the degree of model changes and increase those changes' correctness.

Research Design

We conducted an experiment to empirically test our claims. The independent variable was the modeling approach. The dependent variables were the syntactic and semantic accuracy of the participants' model comprehension and the degree and correctness of the changes.

The participants were 64 senior undergraduate IT students in system analysis and design courses that included advanced UML training. The participants had previously taken at least two object-oriented courses and one UML course. They also received training and completed exercises on Enterprise Mobile Application DSM (EMADSM), a domain-specific language developed for the mobile phone industry.[2] Although the amount of training on EMADSM was much shorter than on UML, the subjects had sufficient levels of proficiency with both approaches.

The experimental task involved designing a mobile-phone application for conference registration. The implementation target was a Symbian S60-based mobile-phone application framework for enterprise applications. The experimental material consisted of a DSM diagram and a set of UML diagrams representing the system design at the design phase's end. The UML schema included a use case diagram, three sequence diagrams, three activity diagrams, and a class diagram. The DSM design schema represented the same information in EMADSM.

We randomly split the participants into DSM and UML groups. We gave them a high-level textual description of the system objectives and requirements and asked them to perform the maintenance task, which involved modifying the models to satisfy a new requirement for the application. After performing the task, the participants answered questions evaluating their syntactic and semantic comprehension and the models' changeability.

Discussion

The results in Table 1 and Figure 1 indicate that the subjects performed better with DSM for each dependent variable. All differences are statistically significant.

Both syntactic and semantic model comprehension were significantly better with DSM. This is especially noteworthy because the subjects had extensive UML training but only brief experience with DSM. Most syntactic errors in

DSM occurred because the participants mistook one component for another. The difference in performance for semantic comprehension was particularly pronounced. We believe that mapping the model to the problem domain was easier in EMADSM because the semantic gap between the model and the problem was smaller.

The DSM models' correctness score was about 20 percent higher than the UML models' score. Most subjects using EMADSM changed models by using the correct components, although some of them incorrectly specified the flow of actions. In contrast, subjects using UML changed their models primarily by adding or modifying design elements (for example, methods) or adding new diagrams. Several subjects neglected to change important components in some UML diagrams, which made the diagrams inconsistent. This finding amplifies the findings for comprehension, because it's easier to understand the design and incorporate changes more accurately with DSM. This results in less maintenance effort and in maintenance that's less likely to generate ripple effects and new bugs. We speculate that this difference is partly because DSM models are more compact.

Finally, the degree of changes in DSM is much smaller than in UML; UML diagrams required nearly twice the number of steps.

As we mentioned before, the participants had significant experience with UML but only brief training with DSM. We believe that the performance improvements with DSM would be even more dramatic if the participants possess similar levels of experience in both modeling languages.

The study has significant practical implications. First, domain experts who aren't technical specialists might be able to improve productivity by using DSM in maintenance tasks. Furthermore, our findings suggest that DSM lets users spend less time understanding implementation or language issues and more time modeling the solution. Users can build DSMs incrementally and easily change them, and in many cases executable applications can be built automatically from these models. So, their use in domains with constantly evolving requirements will likely be beneficial.

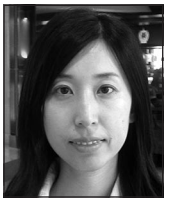
The findings are even more compelling for system maintenance. Because comprehending and modifying DSM models is significantly easier, system maintenance should be faster and cheaper. This, combined with automatic code generation, promises far shorter release cycles and lower costs for providing new features.

We hope that other developer groups will study DSM performance in different application domains to confirm our findings.

References

1. J. Greenfield and K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, John Wiley & Sons, 2004.
2. S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*, Wiley-IEEE CS Press, 2008.

About the Authors



Lan Cao is an assistant professor of information technologies and decision sciences at Old Dominion University. Her major research interests are agile software development and software process simulation and modeling. Cao has a PhD in computer information systems from Georgia State University. Contact her at lcao@odu.edu.

Balasubramaniam Ramesh is the Board of Advisors Professor of Computer Information Systems at Georgia State University. He studies requirements engineering and traceability, agile software development, and knowledge management. Ramesh has a PhD in information systems from New York University's Stern School of Business. He's a member of the IEEE, ACM, and Association for Information Systems. Contact him at bramesh@gsu.edu.

Matti Rossi is a professor of information systems at the Helsinki School of Economics. His major research interests are domain-specific modeling and design research. Rossi has a PhD in information systems from the University of Jyväskylä. He's a member of the ACM and Association for Information Systems. Contact him at matti.rossi@hse.fi.