Perceptions of Software Modeling: A Survey of Software Practitioners

Andrew Forward[1], Timothy C. Lethbridge[1], Omar Badreddin[1]

[1]School of Information Technology and Engineering
University of Ottawa, Canada
+1 613 236-6240, +1 613 562-5800 x6685, +1 757 644-4494
{aforward, tcl} @site.uottawa.ca, obadr024@uottawa.ca

Abstract. In this paper we analyze the results of a survey on how, when and why some software developers model, and why many prefer not to model. The survey of 113 software practitioners studied the reasons developers chose code-centric versus model-centric software engineering, and also gathered data about the notations and tools used. Key findings include: UML is confirmed as the dominant modeling notation; modeling tools are primarily used for documentation and up-front design with little code generation; modeling tools are also used to transcribe models from other media including whiteboards; and the type and quality of generated code is one of the biggest reported problems.

Keywords: Software modeling, survey, questionnaire, empirical study.

1. Introduction

In most other engineering domains modeling is usually perceived as a core activity. One might expect the same in software engineering, but it seems the biggest area of variability in practice today is the extent to which models are used. France and Rump provide an excellent overview [10] of the current state of the art in modeling.

We present the results of a survey of 113 software practitioners. The objectives include understanding: the perceived difficulties of model-centric, or code-centric software development; the current landscape of how, when and why we model in the hopes of uncovering clues as to why modeling is not universally practiced.

Proponents of modeling and modeling languages would have us believe it is obvious that most of us should be modeling, citing its successes and the obvious benefits [12]. However, many developers toil primarily on vast volumes of source code, rarely looking at diagrams or models, let alone creating or editing them. This is typified by the open-source community where file patches are the key unit exchanged [11]. If modeling is so great, why are these people not doing it more?

There has been a small amount of research into why modeling is not that prevalent. Afonso et al [1] point out that "there is little practical evidence of the impact" of model-driven development in general software engineering. Berenbach et al [6] describe common modeling problems that lead to models being less effective, and less adopted. These include a belief that it is just about "pretty pictures", not understanding the underlying paradigm well enough, and lack of attention to consistent style. Anda et al. [4] studied modeling in a safety-critical context, where it

would seem particularly appropriate. They reported that modeling yielded positive results, but the tool inadequacies and training costs were important obstacles. Other obstacles they cited were difficulty applying modeling in a legacy environment, and management processes for developer code ownership. In both these cases the code-centric approach is presupposed.

Dobing and Parsons [7] conducted a web-based survey (supported by the OMG and received 171 responses) on how UML is used in practice. Among their results are that class diagrams, although being the most frequently used UML diagram type, are "not well understood" by 50% of analysts. The majority of respondents found that all aspects of UML are useful for most projects. The authors suggest that complexity and lack of usage guidelines are the biggest concerns with UML.

Several researchers have conducted controlled experiments to measure the benefits of modeling; but the results have not been encouraging. Arisholm et al. [5] concluded that the costs to maintain UML documentation in the software they studied only balanced its benefits. This is echoed by Agerwal et al, [3], [2] who conducted usability experiments of modeling tools. Sjoeberg et al [14] provide a comprehensive survey of experiments, some of which relate to modeling.

Reasons for the lack of adoption of modeling seem to boil down to: a) weaknesses in tools, modeling languages and processes; b) the lack of education or training of developers or their managers; and c) satisficing, wherein the benefits of modeling are judged, rightly or wrongly, to not warrant the costs, given the level of quality desired.

2. Method

The survey was conducted online with targeted requests sent to contacts in several organizations and online forums. We discuss the effectiveness of our sampling procedure in Section 6. The survey consisted of 18 questions. Most of these involved several sub-questions answered using 5-point Likert scales. Responses were in ranges such as strongly disagree to strongly agree, or never to always.

The survey asked, Q1: what is or is not a model? Various options were presented and our objective was to see if participants had a preconceived notion about model. Q2-5: how and when do you model, and using which notations? The objective of these questions was to understand the state of the practice. Q6: how practitioners approach a new task or feature. Q7-10: tools, methods and platforms participants use. Q11-13: do you use modeling, and how good is it. Q14-16: how do code-centric and model-centric approaches compare, and what obstacles do they face. Q17: open-ended free form question for comments from the participants.

Randomization of the questions helped reduce bias towards either code-centric or model-centric questions. Of the 113 participants, at least 88 answered each question, and at least 63 answered the two modeling tools questions. Participants were able to ignore these questions if they did not have substantial experience with modeling tools. A technical report of the questions and complete results is available at [8].

3. Demographics

Participants averaged 14 years experience (80% > 5 years, 20% > 20 years). About two thirds were from Canada or the USA. The rest of the world was fairly well

represented with participation from the United Kingdom, the rest of Europe, India, and Pakistan, as well as a few participants from Australia, Mexico and Singapore.

The most prevalent type of software that participants work on is business software (46% at least 'very often'), design and engineering software (25%), and website content management (23%). The least represented were malware (2%), industrial control (10%), and system utilities (7%). At least one participant had 'always' worked in one of the available categories (except for malware). The dominant programming language in use was Java (60% at least sometimes, 37% often). About 50% use languages like PHP and Perl, and 20% use C/C++ at least sometimes. Modeling is performed at least sometimes by over 95% of participants (57% very often / always). Conversely, 86% at least sometimes work with source code, and 49% very often or always work with source code.

We used Student's t test to determine statistical significance being sub-samples where 90% confidence is $|t| \geq 1.64$, 95% confidence is $|t| \geq 1.96$, and 99% confidence is $|t| \geq 2.58$.

4. Results

4.1 What is a software model?

We asked: "To what extent do you consider the following to be a model of a software system?" We wanted to learn what practitioners believe modeling to be, before their perception was perhaps modified by later questions.

Table 1: Responses for Question 1: What is a Model?

Entity that might be a model	N	% Str. Disagree (1)	% Disagree (1 + 2)	% Agree (4 + 5)	% Str. Agree (5)
Class Diagram	112	0.9	2.7	88.4	48.2
UML Deployment Diagram	111	1.8	5.4	77.5	36.0
Use Case Diagram	112	1.8	9.8	82.1	33.9
Picture By Drawing Tool	111	1.8	7.2	85.6	25.2
Textual Use Case	113	2.7	10.6	78.8	30.1
Picture By Hand	113	4.4	8.8	78.8	29.2
Whiteboard Drawing	112	3.6	9.8	57.1	22.3
Source Code	111	13.5	38.7	46.8	23.4
Source Code Comment	112	11.6	41.1	33.9	9.8

Values range: Strongly Disagree (1), Disagree (2), Neutral (3), Agree (4), to Strongly Agree (5).

In many tables in this paper we provide the following data: 'N' is the number of participants responses. The next four columns highlight the percentage that strongly disagree (lowest rating of 1), disagree (1 or 2), agree (4 or 5), or strongly agree (highest rating of 5) to the particular statement. This approach gives a sense of any potential polarization of the data. The data is sorted based on the mean and does not reflect the order in which the sub-questions were asked.

As Table 1 shows, participants almost completely agree that class diagrams, use case diagrams and UML deployment diagrams all represent models of a software system. In fact, hardly anybody felt that any of these artefacts are not models. There was good agreement that the following can be models: textual use cases, white board drawings, pictures created from drawing programs, and pictures created by hand.

A key conclusion from this question is that developers do not limit their perception of a model to diagrams in a modeling tool. Models can be represented textually (use cases, and source code), as outside of an electronic format (white boards, and hand drawn pictures), and in drawing tools.

4.2 How do the participants create or maintain their models?

We asked, "To what extent do you create or modify software models or modeling information in the following ways?"

Many participants (45% very often) create models using a whiteboard. Other approaches include diagramming tools (37% very often, 42% sometimes), and word-processing software (27% very often, 46% sometimes). Modeling tools were identified as the third least used method of creating a software model (22% very often, and 51% sometimes).

When comparing the results of how the participants model, there are several differences among the sub-samples. The most interesting differences are highlighted below:

- Participants outside Canada and the USA are more likely to use modeling tools (t=2.09) compared to the entire sample.
- Those who actually develop software are less likely to use modeling tools compared to the entire sample (t=-2.46).
- Software developers in Canada/USA are more likely to use a whiteboard compared to software developers outside Canada/USA (t=2.54)
- Software modellers that generate code are less likely to use word of mouth (t=-2.21), whiteboard drawings (t=-1.98) and drawing software (t=-2.63) to model, compared to software modellers that do not generate code

4.3 What sources of information do participants refer to?

We asked, "To what extent do you refer to the following sources of information when you want to learn about the design of a software system?" We explicitly avoided the use of the term 'model' in this question. We wanted to learn the extent to which people refer to the more formal types of models; i.e. in modeling tools or traditional CASE tools, vs. material formulated in less formal ways.

The most important source of design information was word of mouth (55% very often, 22% sometimes). The next most important sources were word processors (48% very often, 30% sometimes), diagramming tools (42% very often, 32% sometimes), and whiteboards (42% very often, 35% sometimes). Least important was fully integrated modeling tools (20% very often, 56% sometimes).

The contrast in answers from the previous question (what tools do you model with) suggest that people responsible for creating software designs should be aware that the medium to create models sometimes differs from the best way to disseminate that information.

The following are a few pertinent observations when comparing subsamples.
- Participants outside Canada/USA are more likely to refer to modeling tools (t=2.19) for information compared to the entire sample.
- Software modellers that generate code are less likely to refer to word of mouth (t=-3.50), and whiteboard drawings (t=-1.99) compared to those that do not generate code.

4.4 When do participants visually document a design?

We asked, "At what point(s) in time do you visually document a design?". We wanted to learn whether developers follow the recommended practice of modeling before coding, or follow an alternative practice. In this question we were careful to include all forms of visual media.

Most practitioners that model do indeed model before writing code (60% very often, 19% sometimes). Few document after writing code (20% very often, 60% sometimes). Only 10% said that they only visually document on request. This is interesting because it shows that a non-trivial fraction of software practitioners do not perceive much value in visual notations as an aid to their day-to-day activities.

4.5 What modeling notations do participants use?

We asked, "To what extent do you use the following notations for the purpose of modeling or design (if you don't know what one of these is, then ignore that particular item)".

As Table 6 shows, about 52% of practitioners very often use to always use UML - and more practitioners use UML 2.* versus UML 1.*. UML is also indicated as being the most highly used notation.

Table 2: Responses for Question 5: What modeling notation do you use?

Language used to model	N	% Never (1)	% Sometimes (1 + 2)	% Very Often (4 + 5)	% Always (5)
UML (any version)	110	18.2	30.9	51.8	22.7
UML 2.*	96	30.2	52.1	34.4	12.5
SQL	108	30.6	55.6	29.6	10.2
Structured Design models	102	19.6	58.8	21.6	9.8
UML 1.*	93	38.7	54.8	28.0	7.5
ERD	106	33.0	63.2	20.8	10.4
Well-defined DSL	104	54.8	78.8	5.8	1.0
ROOM / RT for UML	99	69.7	85.9	7.1	2.0
SDL	93	80.6	89.2	3.2	0.0
Formal (e.g. Z, OCL)	99	78.8	93.9	2.0	1.0

Values range: Never (1), Sometimes (2), Moderately often (3), Very often (4), to Always (5).

Our survey supports the idea that the UML is, in fact, now the universally accepted notation for software modeling as few people are using domain-specific languages for their projects, or other notations like Entity-Relationship Diagrams (ERD), SQL and Specification and Description Language (SDL).

Other notations indicated in use by the participants include: pseudo-code, unstructured drawings (hand-drawn boxes, rough drawings of data structures and processes), Message Sequence Charts, User Requirements Notation (UCM + GRL), BPML (Business Process Modeling Language from SAP), BPMN (Business Process Modeling Notation, in-code modeling, and abstract action language. One participant commented: "the code itself is a model, and should speak well".

4.6 Current Use of Modelling Tools

We asked, "To what extent do you use software tools in the modeling process for the following activities?"

Modeling tools are mostly used to design software (48% very often), and used to digitize a *paper* design (39% very often). Few generate some code (which is then edited by hand) from modeling tools (18% very often), and even fewer (14% very often) generate all code from tools.

The following differences surfaced among the sub-samples:

- Real-time developers are more likely to use modeling tools for prototyping a design compared to entire sample (t=2.07)
- Participants that generate code are much more likely to use modeling tools for almost all (except brainstorming) of the cited reasons including transcribing (t=2.04), developing a design (t=3.15), prototyping (t=3.42), generating some code (t=5.52), generating all code (t=4.88) compared to the entire sample.

4.7 Important Attributes of a Software Model

We asked, "Please rank the following attributes of a software model from most important (1) to least important (9)." The results are presented in Table 9, The percentages show how many participants ranked each item in the top 4, and bottom 4.

Table 3: Responses for Question 13: Most important attributes of a modeling tool?

Attribute / Ability to	N	Rank	% top 4	% bottom 4
Communicate to others	89	1	78.7	16.9
Readability	89	2	68.5	20.2
Ease and speed to create	89	3	55.1	36.0
Ability to analyze	89	4	55.1	33,7
Collaborate amongst developers	89	5	43.8	38.2
Ability to view different aspects of a model	89	6	40.4	42.7
Generate code	89	7	23.6	70.8
Information density	88	8	17.0	72.7
Embed parts of model in documentation	89	9	13.5	82.0

The **% top 4** show the % of participants that listed the attribute in their top 4. Similar for **% bottom 4**.

Participants seem to care most about how well a software model is readable and communicate to others. This is unsurprising but the next two attributes are more interesting. The third most important attribute is how quickly and easily models can be created. The fourth most important attribute suggests that modeling tools should

facilitate analysis. Although our survey did not expand on the exact meaning of these attributes, the strong response warrants further investigation.

Software modeling tools should allow a developer to create models quickly and easily. We are not sure if the current tooling provides either benefit. Visual tools do not necessarily provide an intuitive interface to allow for easy development, and the inefficiencies of mouse actions versus keyboard strokes may limit the tools' abilities to be quick and efficient for the developer.

Participants did not place much importance in the ability of models to generate code; this was the 3rd least important attribute. From earlier questions we saw that most participants do not use the code-generating feature of their modeling tool, and most tools are perceived as doing a poor job of generating code. The fact that this question indicates that code generation is not even considered important might arise simply because developers haven't yet experienced the benefit and power that code-generation can provide. In other words perhaps software developers are not ready for code-generating modeling tools. Alternatively there might be an intrinsic power in being able to work with source code that code-generation enthusiasts do not recognize. Or maybe, what is really needed is to raise the level of abstraction of programming languages so they incorporate the features found in models, rather than generating low-level code from the visual models. These issues are a subject of our future research.

4.8 Code-Centric versus Model-Centric Development

Software development can be approached from various perspectives:
- Model-only: The model is effectively all there is, except for small amounts of code.
- Model-centric: Modeling is performed first and code is generated from the model, for possible subsequent manual manipulation.
- Model-as-design-aid: Modeling for design purposes, code mostly written by hand.
- Model-as-documentation: Modeling to outline or describe the system, largely after the code is written.
- Code-centric: Modeling is almost entirely absent.

As is observed in the previous section reporting the current use of modelling tools, the results of our survey do contain practitioners that fit each level of this spectrum shown. In the following questions we analyze how well modeling tools perform specific tasks as well as uncover what approach (model versus code) is more suitable based on particular situations.

The following questions make reference to model-centric and code-centric approaches to developing software. Participants were provided with an explanation of both approaches prior to being presented with these questions.

4.9 Which Approach: Model vs. Code?

We asked, "For each of the following, how do code-centric development approaches compare to model-centric approaches." The results are presented in **Table 4**. The percentages indicate those who thought the task was *much* easier in each approach.

Table 4: Responses for Question 14: Tasks that are better in a model-centric

Available activities	N	mean	s.d.	% Much easier in Models (1)	% Easier in Models (1 + 2)	% Easier in Code (4 + 5)	% Much easier in Code (5)
Fixing a bug	90	3.2	1.5	21.1	28.9	43.3	25.6
Creating efficient software	92	3.1	1.4	16.3	35.9	43.5	21.7
Creating a system as quickly as possible	92	3.0	1.5	23.9	46.7	42.4	23.9
Creating a prototype	92	2.9	1.5	26.7	43.0	32.6	22.8
Creating a usable system for end users	92	2.7	1.3	26.1	42.4	22.8	10.9
Modifying a system when requirements change	91	2.5	1.4	34.1	54.9	24.2	13.2
Creating a system that most accurately meets requirements	91	2.2	1.3	42.9	67.0	19.8	8.8
Creating a re-usable system	92	2.2	1.3	44.6	63.0	15.2	9.8
Creating a new system overall	92	2.2	1.3	43.5	68.5	20.7	7.6
Comprehending a system's behavior	89	2.0	1.3	51.7	71.9	15.7	5.6
Explaining a system to others	92	1.7	1.1	61.1	81.8	7.6	6.5

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

The results suggest that overall, most activities tend to be easier in a model-centric approach, in the opinion of participants. Even the task judged to be the most code-centric (creating efficient software) was considered to be achievable with about the same amount of difficulty as in a model-centric approach. However, the results to this question overall reflect considerable polarization of the participants, since even on the most model-centric activities, around 10% believed that the code-centric approach was much easier than a model-centric approach.

Model-centric approaches appear to be particularly appropriate for higher-level activities including: creating a re-usable system, a system that meets requirements, understanding a system and explaining a system to others.

The participants seem to really want to incorporate modeling into their process, but as earlier questions indicate, at present they are not doing so.

4.10 Problems with the Model-Centric Approach

We asked, "Which of the following are potential difficulties with modeling. These may be reasons why you don't model much, or things you find hard about modeling." The results are in **Table 5**.

The biggest problem of model-centric approaches is perceived to be keeping the model up to date with the code (recall that few participants want code to be generated from models). Model-centric tools may benefit from features that help to better synchronize code and models; provide better traceability between the two; or, even provide better modeling capabilities within the source code itself to reduce the need for external, disjoint, or duplicate modeling artefacts.

The following differences surfaced among the sub-samples:

- Extensive programmers are more likely to agree that modeling tools are too 'heavyweight' (t=1.69), and more likely to agree that the code generated from modeling tools is 'not of the kind I would like' (t=1.79)
- Programmers that model are less likely to agree that modeling languages are hard to understand (t=-2.07) and do not provide enough detail to be implemented (t = -1.80), but more likely to agree they become out of date / inconsistent with code (t=1.74)
- Experienced modelers (> 11 years experience) are less likely to agree that models do not provide enough detail to be implemented (t=-1.85), and that modeling tools change so models become obsolete (t=-1.76)
- Modellers that generate code are less likely to find the resulting code "of the kind I would like" (t=-1.92) compared to modellers that do not generate code.

Table 5: Responses for Question 15: Problems with a model-centric approach.

Potential problems	N	mean	s.d.	% Strongly Disagree (1)	% Disagree (1 + 2)	% Agree (4 + 5)	% Strongly Agree (5)
Models become out of date with code	92	3.8	1.2	7.6	16.3	68.5	37.0
Models cannot be easily exchanged	91	3.3	1.3	15.4	26.4	51.6	17.6
Modeling tools are 'heavyweight'	92	3.1	1.2	10.9	31.5	39.1	12.0
Code generated from a modeling tool not of the kind I would like	91	3.0	1.4	18.7	39.6	38.5	16.5
You cannot describe the kinds of details that need to be implemented	89	2.8	1.3	23.6	43.8	36.0	7.9
Creating and editing a model is slow	92	2.7	1.2	17.4	43.5	22.8	12.0
Modeling tools change, models become obsolete	92	2.7	1.2	22.8	44.6	32.6	5.4
Modeling tools lack features I need	89	2.6	1.1	19.1	44.9	21.3	5.6
Modeling tools hide too many details that would be visible in the code	92	2.6	1.1	19.6	44.6	23.9	1.1
Modeling tools are too expensive	90	2.6	1.3	26.7	46.7	26.7	6.7
Modeling tools do not allow me to analyze my design how I would want	90	2.5	1.3	28.9	51.1	25.6	6.7
Org. culture does not like modeling	92	2.5	1.2	31.5	48.9	23.9	4.3
Semantics of models different from those of programming Languages	90	2.4	1.3	31.1	56.7	23.3	8.9
Modeling languages are not expressive enough	91	2.4	1.1	28.6	54.9	17.6	2.2
Modeling language hard to understand	91	2.2	1.0	28.6	62.6	9.9	3.3
Had bad experiences with modeling	91	2.2	1.2	39.6	63.7	16.5	6.6
Do not trust companies will continue to support their tools	89	2.0	1.0	44.9	67.4	10.1	0.0

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

Participants did not indicate past bad experiences with modeling which suggests little bias against model-centric approaches. Also, modeling languages are not the limiting factor to adopting model-centric approaches, since languages are not considered that difficult to understand. The top fear of participants is that models tend to become out of date. This supports earlier discussion where we pointed out that

tools need to focus on synchronization. However, if developers generated all code using a tool, this would cease to be a problem!

Another high ranking problem is that "code generated from a modeling tool is not of the kind I would like". The message from developers seems to be that even in environments where code is fully generated, they would be comfortable only if the code is easy to understand. Perhaps they feel that be ability to edit code or to debug it is a necessary safety net until full generation of code becomes the norm.

It is interesting to note that 34% felt that a model's underlying storage format would become obsolete (5th most perceived problem). Text and source code seem to have a much longer shelf-life – maybe it is time to explore storing models in a textual manner that is human editable just like a programming language. This theme that is resonated throughout this paper and is supported by results from multiple questions.

4.11 Problems with the Code-Centric Approach

We asked, "Which of the following are potential difficulties with code-centric development (i.e. lacking modeling)." The results are presented in Table 6.

Table 6: Responses for Question 16: Problems with a code-centric approach.

Potential problems	N	mean	s.d.	% Strongly Disagree (1)	% Disagree (1 + 2)	% Agree (4 + 5)	% Strongly Agree (5)
Hard to see overall design	94	3.8	1.1	4.3	13.8	66.0	35.1
Hard to understand behavior of system	94	3.6	1.1	4.3	19.1	60.6	21.3
Code quality degrades over time	92	3.4	1.3	9.8	28.3	55.4	25.0
Too difficult to restructure system	93	3.4	1.2	8.6	22.6	51.6	17.2
Buggy code changes	93	3.4	1.2	9.7	22.6	50.5	18.3
Changing code takes too much time	94	2.8	1.2	20.2	39.4	27.7	8.5
Our language leads to complex code	94	2.5	1.2	26.6	51.1	20.2	8.5
More skill is required than is available to develop high quality code	91	2.5	1.2	29.7	53.8	22.0	6.6
Our languages are not expressive enough	91	2.1	1.2	46.2	64.8	14.3	5.5
Organization culture does not like the code-centric approach	92	1.9	1.2	58.7	72.8	14.1	4.3
Our programming language is likely to become obsolete	93	1.9	1.1	51.6	75.3	9.7	3.2

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

Participants feel that code-centric approaches fail to deliver a high-level view of the system, and entropy means the situation only gets worse over time. Participants are of the opinion that programming languages they use do not result in complex code, are expressive enough and unlikely to become obsolete. The participants were divided if changing the code takes too much time; 35% agree and 36% disagree.

The following differences surfaced among the sub-samples:
- Software developers are less likely to see that changing code takes too much time (t=-220), or that a system is too difficult to restructure when needed (t=-2.53) compared to the entire sample.
- Software developers that also model are more likely to see code becoming of poorer quality over time (t=2.34) compared to software developers that do not model

- Practitioners that generate code are more likely to see programming language code becoming obsolete (t=2.05) and that programming languages are not expressive enough (t=2.11) compared to the entire sample.

5. Participant Additional Comments

The following is a sample of the comments provided by the participants in answer to an open-ended question that asked for comments about the "the pros and cons of modeling, or your experiences regarding the topic of this survey".

- *Adding too much detail to a model takes too much time, and can potentially confuse developers when it comes to implementation.*
- *Modeling using a tool is good for documenting a model, but otherwise a piece of paper/whiteboard works better and is more flexible.*
- *Modeling using a tool is good for documenting a model, but otherwise a piece of paper/whiteboard works better and is more flexible.*
- *Code wins over models every time when it comes to revenue. Working, tested code with business value can be sold.*

6. Threats to Validity

Below we have outlined the steps we took to mitigate the main threats to validity.

Question interpretation. The survey was conducted over the Internet and respondents may have misunderstood our questions. To reduce ambiguity we reviewed the survey with colleagues, and conducted trial runs with team members.

Researcher bias. The survey attempts to uncover problems with model-centric and code-centric approaches to software development. A potential bias could be introduced if our survey appeared to be overly negative towards either approach. To reduce the chance of bias we tried to be objective when referring to both code-centric and model-centric questions, as well as presenting the questions in a random order.

Non randomized sample. To help ensure a representative collection of software practitioners we approached open and closed forums. We submitted articles to popular technology sites Digg.com and Dzone.com. We submitted emails to UML, Agile, Java and RUP user groups. We submitted personal requests to colleagues. Our demographics results indicate that we do have representation from most regions of the world, educational backgrounds, software industries, and types of developers.

Missing/inappropriate question options. Most questions were closed without an option for participant comments. A closed questionnaire can improve participation rates, and is easier for quantitative analysis, but it can limit the kind of feedback received. It is almost impossible to provide all scenarios to explain a certain occurrence. Too much freedom in the data collection process reduces the quantitative abilities of the survey. To mitigate this threat the survey was reviewed with colleagues; and concluded with a free-form open question for user comments.

7. Conclusions

We reported on a study of 113 software practitioners from around the world. We were motivated to better understand the resistance to modeling, and to uncover clues to

overcome this resistance. Most participants take a broad view of what modeling is; and some agree that even hand-drawn diagrams can be considered a model. Our results also show that the use of formal modeling is not widespread. UML is a dominant modeling language, but it tends to be used informally.

For those who use modeling tools, the main uses are to develop or digitize a design. It is uncommon to generate all code from these tools. Our data suggest this is because the generated code is not of the type or quality they want. Overall, most developers clearly see the value of the modeling approach, even though they only practice it to a limited degree. But, there remains a hard core of code-centric 'zealots' who seem dead set against modeling. Several questions lend support to the hypothesis that uptake of modeling might be higher if it were possible to create models textually, in the same manner that code-centric developers currently work. If this was possible, then the complaints our participants had about the quality of generated code might be reduced, and uptake of modeling as a whole might increase. To explore these ideas our team is working on a technology to unite code-centric and model-centric development; more information is available at [9].

References

[1] Afonso, M., Vogel, R., Teixeira, J., 2006, "From code centric to model centric software engineering: practical case study of MDD infusion in a systems integration company", Int. Wkshps. on Model-Based Development of Computer-Based Systems, 2006, IEEE Computer Society, 10 pp.

[2] Agarwal, R. and Sinha, A. P., 2003, "Object-oriented modeling with UML: a study of developers' perceptions", CACM 46, 9 (Sep. 2003), PP. 248-256.

[3] Agarwal, R., De, P., Sinha, A. P., and Tanniru, M, 2000. "On the usability of OO representations". CACM 43, 10 (Oct. 2000), pp., 83-89

[4] Anda, B., Hansen, K., Gullesen, I., and Thorsen, H.K., 2006, "Experiences from introducing UML-based development in a large safety-critical project", ESE, 11, 4, Dec. pp 555-581

[5] Arisholm, E, Briand, L.C., Hove, S.E. and LaBiche, Y., 2006 "The impact of UML documentation on software maintenance: an experimental evaluation", IEEE TSE, 32, 6, June, pp. 365-381.

[6] Berenbach, B, and Konrad, S., 2007, "Putting the "Engineering" into Software Engineering with Models", MISE'07, IEEE Computer Society, pp 4-4

[7] Dobing, B and J. Parsons, 2006, "How UML is Used", CACM, 49, 5, (May 2006), pp. 109-114.

[8] Forward, A. TR-2008-07 and TR-2008-08: Modeling survey data and software application taxonomy data available at http://www.site.uottawa.ca/eng/school/publications/techrep/2008/

[9] Forward, A. Lethbridge, T. C. Improving Program Comprehension by Enhancing Program Constructs: An Analysis of the Umple language. ICPC 2009

[10] France, R, and Rumpe, B, "Model-driven Development of Complex Software: A Research Roadmap", FoSE, ICSE 2007, pp. 27-54.

[11] Hertel, G., Niedner, S., and Herrmann, S., 2003, "Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel", Research Policy 32, pp. 1159-1177.

[12] Lavaggno, L., Martin, G., and Selic, B., eds., UML for Real: Design of Embedded Real-Time Systems, Springer, 2004

[13] Procaccino, J, Verner, J, Shelfer, K, and Gefen, D., (2005) "What do software practitioners really think about project success: an exploratory study", J. Systems and Software, 78 pp.194–203, 2005

[14] Sjoeberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.-K., Rekdal, A.C , 2005., "A survey of controlled experiments in software engineering", IEEE Trans. SE, 31, 9, Sept 2005, pp. 733-753/

[15] Sultan F. and Chan, L., 2000, "The adoption of new technology: the case of object-oriented computing in software companies", IEEE Trans. Engg. Mgmt. 47, 1 pp 106-126.