Problems and Opportunities for Model-Centric vs. Code-Centric Development: A Survey of Software Professionals

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. We present some results of a survey of 113 software practitioners conducted in 2007. The aim of the survey was to uncover their attitudes and experiences regarding software modeling, as well as development approaches that avoid modeling. We were motivated by observations that modeling is not widely adopted; since many developers continue to take a code-centric approach. A key finding is that modeling tools are primarily used to create documentation and for up-front design with little code generation. Participants also believe that model-centric approaches to software engineering are easier but are currently not very popular since most participants currently work in code-centric environments. A finding from sub-sampling is that problems identified with model-centric approaches are similar regardless of a participant's country. Also, programmers that model extensively (versus those that do not model much) are more likely to agree that models become out of date and inconsistent with code.

Keywords: Software modeling, survey, questionnaire, software practitioners, empirical study, attitudes.

1. Introduction

One of the biggest areas of variability in software engineering practice today is the extent to which a development team uses models. At one extreme, model-centric or model-driven development, software engineers use tools to describe the structure and behavior of their system using a language like UML, and then generate source code for the software automatically. France and Rump provide an excellent overview 9 of the current state of the art in modeling. At the other end of the spectrum are those developers who work entirely by editing code, i.e. using a code-centric approach.

Several researchers have conducted controlled experiments to measure the benefit that modeling may provide; however the results have not been encouraging for the modeling community. For example Arisholm et al. 5 concluded that the costs of maintaining UML documentation in the type of software they studied balance the benefits of the modeling. This sentiment is echoed by Agerwal et al, 3, 2 who

conducted experiments to examine the usability of modeling tools. Sjoeberg et al 13 provide a comprehensive survey of experiments, some of which relate to modeling.

In this paper we will highlight some of the results uncovered from a survey of 113 software practitioners. Our objective for conducting the survey was to understand what makes developers take a model-centric versus a code-centric approach. We wish to use that knowledge to improve both modeling and coding tools, and perhaps to help bridge the gap between them.

Our survey complements the survey results of Dobing and Parsons 7 in that we asked questions about modeling in general, as opposed to specifics of UML.

2. Method

The survey was conducted online 8. We sent targeted requests to personal contacts in a wide variety of organizations. We also asked for participation using a variety of Internet forums.

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 was divided into groups of questions as follows:

- Q1: What is or is not a model? Various options were presented ranging from class diagrams, use cases, to source code. Our objective was to see if participants had a preconceived notion about what a model is.
- 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 do you approach a new task or feature with respect to requirements, design, modeling, testing and documentation?
- Q7-10: What tools, methods and platforms do you use, and what type of software do you develop?
- Q11-14: To what extent do you use modeling, and how good is it for various tasks.
- Q15-16: What are the principal difficulties you perceive with the model-centric and code-centric approaches?
- Q17: An open-ended free form question for comments about the survey and / or modeling in general.
- Q18: Demographics question with sub-questions about country of origin, education level, and years of experience of the participant.

Some randomization of the order of question was applied to reduce bias towards either code-centric or model-centric questions. Questions 2 to 5 were presented in a random order. We then defined modeling as follows so that participants could consistently answer subsequent questions:

> *For the remainder of the survey, please assume that any reference to a software model refers to an artifact that represents an abstraction of the software you are building. A model can typically be viewed as a set of*

diagrams and/or pieces of structured text. It can be recorded on a white board, paper, or using a software tool. A model could use formal syntax and semantics but this is not necessary. We will consider the final source code of the system, and requirements written in natural language to not be models, although models can be embedded in a requirements document.

Questions 7 through 16 were presented in a random order. The survey had 113 participants. Of those, 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. The full technical report outlining the questions posed, method, threats to validity and the results obtained is available in 8.

3. Demographics

Participants averaged 14 years of experience, with 80% having more than 5 years of experience, and about 20% having more than 20 years of experience.

About two thirds of the respondents were from Canada or the United States. 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 (identified as 'very often' by 46% of participants), followed by design and engineering software (25%), website content management and information display such as search, maps and news sites (23%). The least represented categories were malware (2% - good!), industrial control (10%), and system utilities (7%). At least one participant had 'always' worked in one of the available categories (except for malware).

About 90% of participants have leadership roles (team leader, project manager) at least sometimes, and 53% lead very often or always. Design or modeling is performed at least sometimes by over 95% of participants, and 57% do this very often or always. On the other hand, 86% at least sometimes work with source code (developing new code, maintaining and/or bug fixing), and 49% very often or always work with code.

Almost half of the participants had a Masters degree (44%) – which is more than in the general software engineering population. However, an advantage of this over-representation is that it helps ensure our respondents are among the more knowledgeable of software engineers.

In our analysis, we not only studied the whole sample, but also sub-sampled according to various criteria to ascertain whether particular groups of developers had different opinions; the sizes of the major sub-groups are shown in Table 1.

Table 1: Sub-Sample Sizes

Category	N	%
All Participants	113	100%
Participants in Canada/USA	63	56%
Participants Outside Canada/USA*	27	24%
Software Developers	53	47%
Software Modelers	46	41%
Participants that Generate Code	15	13%
Experienced Participants (≥ 12 years)	53	47%
Participants in Real-Time Projects	19	17%

*Of the 113 participants, 23 participants did not indicate what country they live in and were not included in the geographic sub-sampling

For example, we separately analyzed the data for those who reported they programmed extensively, or used modeling tools extensively. We also sought any geographic differences, and differences based on the type of software the developers focused on. Below we have highlighted a few of the more interesting findings based on the entire sample, as well as outlining significant differences (or similarities) among subgroups. We used Student's t test to determine statistical significance, 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 Creating vs. Consuming Software Models

There is little agreement among participants regarding how software models are created and maintained. Similarly, there is also little agreement about among participants about how they learn about the design of software.

The most frequent way of creating models was by drawing or writing on a whiteboard, with 45% agreeing they do this, and 33% disagreeing. Next most frequent was using diagramming tools (37% agree, 42% disagree), and word-processing software (27% agree, 46% disagree). Other mechanisms to maintain software models like word of mouth, handwritten material, source code comments, and modeling tools/CASE had between 22% - 30% agreement that they were used by the participants. Finally, very few (13% agree, 72% disagree) used drawing software to maintain models.

It is interesting to reiterate that that 27% of people feel they create models by word of mouth, although 42% of people disagree with this.

The most important source of design information was word of mouth, with 55% of people very-often or always using information originating from this source, whereas

24% of people sometimes or never use this. The next most important source was material created in word processors (48% very often, 30% sometime), and using diagramming tools (that can create structured diagrams, but not integrated models with 42% very-often and 32% sometimes). Diagramming tools and whiteboards were cited as being used very often by 42% of the participants.

The least important source of information was material in fully-integrated modeling tools: only 32% use material created with such tools very often, and conversely 33% never do. Handwritten material was claimed to be used by 20% very often and never by 24%.

The contrast between what mechanisms participants use to generate models versus consume them is interesting. Although our survey did not attempt to answer why there is a divide, a key conclusion is that the tools and data formats used by individuals responsible for creating software design information tend not to be the same as those used by the audience of that information: In other words, assimilation of data is done differently from dissemination.

4.2 How are you using your modeling tools?

The main uses of modeling tools reported by participants were to develop the design of a software system (48% very often) or to simply transcribe a design into a digital format (39% very often) in somewhat of a data-entry fashion.

Conversely, tools to generate code from the model are not widely used (18% very often generate some code and only 14% very often generate all code for a software system); this may be because participants do not need code generated, or because the tools do not do it in the way they want.

Relatively few participants use software modeling tools to brainstorm about possible design ideas and alternatives (23% very-often, and 55% sometimes). It therefore seems that collaboration amongst developers is not an important feature for software modelers. From earlier questions, it seems that whiteboards are still superior for collaboration and the role of modeling tools in this instance is to help transcribe the design as opposed to actively develop it.

4.3 Code-Centric vs. Model-Centric Approaches

The following questions make reference to model-centric and code-centric approaches to developing software.

In a model-centric approach, the developers look to the model to see the design, and change the model as the first step in performing any design change. Extensive modeling is performed, and the coding is either automated, or at least straightforwardly determined from the model. In a code-centric approach, the code is seen as the main artifact; developers understand the design by understanding the code, and the process of design change is equated with changing the code.

The participants' perceptions about which approach works best for various activities are presented in Table 2.

Table 2: Responses for Question 14: Tasks that are better in a model-centric versus code-centric approach.

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

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

The results suggest that 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, results to this question overall reflect considerable polarization of the participants, since even on the most model-centric activities, between 6% and 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 (63% somewhat easier), a system that meets requirements (67% somewhat easier), understanding a system (72% somewhat easier) and explaining a system to others (82% somewhat easier).

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

4.4 What are the problems with a model-centric approach?

As expected, the biggest problem of model-centric approaches is perceived to be keeping the model up to date with the code (recall that participants did not want code to be generated from models) with 68% in agreement.

Model-centric tools may benefit from features that help to better:

- Synchronize code and models to reduce the inconsistencies.
- Provide better traceability between models and code to help identify relationships among code and model artifacts to help indicate aspects of models that may require maintenance should the code change (and vice-versa).
- Provide better modeling capabilities and expressions within the programming code to reduce the need for external and disjoint modeling artifacts.

The results highlighting the problems with a model-centric approach are presented below in Table 3.

Participants did not identify that they had had past bad experiences with modeling (only 17% agree to have had a bad experience) which suggests little bias against model-centric approaches. UML was identified as the dominant modeling language with 52% usage among the survey participants, whereas the next highest, structured design models, having only 22% usage. Participants found that modeling languages are not the limiting factor to adopting model-centric approaches, since such languages are not considered very difficult to understand (only 10% agree modeling languages are difficult).

It is interesting to note that 34% felt that a model's underlying storage format would become obsolete (this was the 7th most perceived problem). Text and source code seem to have a much longer shelf-life – perhaps it is time therefore to explore better ways to render models in a textual manner that is human editable just like a programming language.

Table 3: 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 and inconsistent with code	92	3.8	1.2	7.6	16.3	68.5	37.0
Models cannot be easily exchanged between tools	91	3.3	1.3	15.4	26.4	51.6	17.6
Modeling tools are 'heavyweight' (to install, learn, configure, use)	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 or want	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 source 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 be to analyze my design in ways I would want	90	2.5	1.3	28.9	51.1	25.6	6.7
Organization 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 used for implementation	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
Have 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

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

4.5 Comparing sub-samples to the entire sample

What follows is a comparison between the entire sample, and various sub-samples for question 15.

- Participants we classify as 'programmers' are more likely to agree that modeling tools are too 'heavyweight' (t=1.69), they are also more likely to agree that the code generated from modeling tools is 'not of the kind I would like' (t=1.79)
- Participants working on real-time systems are more likely to agree that their organizational culture does not like modeling (t=1.80)
- Participants that generate code from models are less likely to agree that modeling tools hide too many details (t=-1.86), and more likely to agree that they do not trust that companies that offer modeling tools will continue to support their tools (t=1.72)

There were no significant differences between participants that model extensively and the entire sample. There were no differences, either, among the participants with a lot of experience (≥ 12 years) or those with little experience (< 5 years); between those living in Canada or the USA versus those living outside Canada and the USA; and between those that only sometimes model versus those that only sometimes generate code from models.

4.6 Comparing between pairs of sub-samples

What follows is a comparison between sub-samples for question 15, looking at the attitudes of modelers (and non-modelers) relative country, experience, type of software application developed, and whether or not the participant is an extensive programmer (i.e. coders) or not.

When comparing participants that are extensive programmers, we compared the responses based on those that model a lot (very-often to always) versus those that model a little (sometimes to seldom). The following differences were observed:

- Programmers that model are less likely to agree that modeling languages are hard to understand (t = -2.07) and are also less likely to agree that models do not provide enough detail to be implemented (t = -1.80).
- Programmers that model are more likely to agree that models become out of date and inconsistent with code (t=1.74)

When comparing experienced participants (≥ 12 years) on the same criteria (those that model a lot versus a little) we observed the following significant differences:

- Experienced modelers are less likely to agree that models do not provide enough detail to be implemented (t=-1.85), and less likely to agree that modeling tools change and models become obsolete (t=-1.76)
- Experienced modelers are more likely to agree that modeling tools are too expensive (t=1.96)

Other points of interest between pairs of sub samples of the participants include:

- Modelers that generate code are less likely to find the resulting code "of the kind I would like" (t=-1.92) compared to modelers that do not generate code.
- Modelers working on real-time systems are more likely to find their organizational culture does not like modeling (t=1.72) compared to those not on real-time systems
- There are no significant differences between modelers in Canada / USA and the rest of the world, as well as modelers with a lot of experience (12 or more years), and those with little experience (fewer than 5 years).

4.7 What are the problems with a code-centric approach?

We asked participants about problems involved with code-centric approaches to software development. The results are presented in Table 4.

Participants feel that code-centric approaches fail to deliver a high-level view of the system (66% agree) and entropy means that the situation only gets worse over time (55% agree).

Conversely, participants are of the opinion that programming languages they use do not result in complex code (20% see this as a problem), are expressive enough (14% see this as a problem) and are not likely to become obsolete (10% see this as problem).

The participants were divided as to whether or not changing the code takes too much time; 27% agree and 39% disagree.

5. Conclusions

The following points are the main findings from our study.

It is clear that most participants take a broad view of what modeling is. They include informal material such as hand-drawn diagrams in what they consider to be a model. The use of formal modeling tools is, however, not widespread, with over a third never using them. UML is the dominant modeling language for modeling – but tends to be used informally.

For those who use modeling tools, the main uses are to develop a design or transcribe a design into digital format. Use of a tool to generate all code is not common; this is seemingly because participants feel that the generated code is not of the type or quality they want, and presumably also because most developers are reluctant to use full-fledged modeling tools capable of generating code.

Table 4: 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 becomes of poorer quality over time	92	3.4	1.3	9.8	28.3	55.4	25.0
Too difficult to restructure system when needed	93	3.4	1.2	8.6	22.6	51.6	17.2
Difficult to change code without adding bugs	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 programming 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
Programming 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

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

Overall, most developers clearly see the value of the modeling approach, even though they practice it only to a limited degree. However, there remains a hard core of code-centric zealots who seem dead set against modeling.

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 and Model-Based Methodologies for Pervasive and Embedded Software, 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”, Empirical Software Engineering, 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 Trans. Softw. Engg., 32, 6, June, pp. 365-381.
6. Berenbach, B, and Konrad, S., 2007, “Putting the "Engineering" into Software Engineering with Models”, Int. Workshop on Modeling in Software Engineering (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. Modeling survey data and software application taxonomy data available at http://site.uottawa.ca/~tcl/gradtheses/aforwardphd/
9. France, R, and Rumpe, B, “Model-driven Development of Complex Software: A Research Roadmap”, FoSE, ICSE 2007, pp. 27-54.
10. 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.
11. Lavaggno, L., Martin, G., and Selic, B., eds., UML for Real: Design of Embedded Real-Time Systems, Springer, 2004
12. 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
13. 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/
14. 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