The Evolution of Software Design Practices Over a Decade: A Long Term Study of Practitioners

Omar Badreddin[*], **Khandoker Rahad**[†], **Andrew Forward**[‡], **and Timothy Lethbridge**[§]
[*]University of Texas, USA
[†]University of Texas, USA
[‡]University of Ottawa, Canada
[§]University of Ottawa, Canada

ABSTRACT We present the results of a survey of 248 software practitioners conducted in three phases ten years apart. The goal of the study is to uncover trends in the practice of software design and the adoption patterns of modeling languages such as UML. The first phase was conducted in April-December 2007 and included 113 responses. The second phase was conducted in March-November 2017 and included 115 responses. The third phase is a post-survey study was conducted in November 2018 and included additional questionnaires with 20 participants. All survey phases were conducted online, employed identical solicitation mechanisms, and included the same set of questions. The survey results are analyzed within each phase and across phases. We present the results and analysis of the data identifying upward and downward trends in design and modeling practices. The results indicate a significant increase in the use of well-defined and formal modeling languages, as well as a marked increase in the adoption of Domain-Specific Languages. This is also reflected in a significant increase in the adoption of forward engineering methodologies. A key motivation for this uptake is a concern that programming languages and platforms may become quickly outdated. Unfortunately, there has been a consistent dissatisfaction with modeling tools features, particularly their ability to support effective communication and collaboration. This is mirrored by increasing dissatisfaction with modeling tools usability and learnability. Future projections of this study suggest that diversity in modeling languages and tools is likely to continue to grow, as well as the increase in reliance on models for automated artifacts generation. As such, model and tool interoperability is likely to become an even greater concern for the years to come. The results of this study can help researchers, practitioners, and educators to focus efforts on issues of relevance and significance to the profession. Specifically, this research will advocate to build better software modeling tools and promote modeling to the educators.

KEYWORDS Software Design, Software Modeling, UML, Practices, Survey.

1. Introduction

The adoption of design in the software engineering spheres has been far from uniform. In domains sensitive to deficiencies such as safety-critical systems, software engineers have unreservedly adopted development practices that ensure exceptionally high levels of reliability and security. Those practices include 1) extensive use of model-driven engineering methodologies where models generate all or most of the executable artifacts, and 2) model-driven testing methodologies where many test cases, scenarios, and test oracles are automatically generated and executed to achieve more efficient testing and enhanced coverage. These approaches enable engineers to verify and validate software systems even in absence of complete or executable code and, in many cases, independently of the platform. Models support advanced simulations of software and physical systems, enabling the testing of rare scenarios that may otherwise be too

JOT reference format:
Omar Badreddin, Khandoker Rahad, Andrew Forward, and Timothy Lethbridge. *The Evolution of Software Design Practices Over a Decade: A Long Term Study of Practitioners*. Journal of Object Technology. Vol. 20, No. 2, 2021. Licensed under Attribution 4.0 International (CC BY 4.0)
http://dx.doi.org/10.5381/jot.2021.20.2.a1

expensive or risky to execute. Model-centered methodologies ensure that potential vulnerabilities in software codes are discovered early and are not obscured by arbitrary code complexities (Hutchinson et al. 2011). It has been argued that dissemination of model-centered methodologies would eventually make its way to the main-stream practices (Anda et al. 2006a).

Despite the near consensus on the value-added of model-centered methodologies (Budgen et al. 2011), many studies have exposed fundamental deficiencies in model-centered approaches (Badreddin et al. 2013). Arisholm et al. (Arisholm et al. 2006) concluded that the major benefits of using modeling in the form of UML diagrams are wiped out by the costs associated with maintaining such models. Iivari (Iivari 1996) explored why modeling tools are not used in general in software projects. In their study of organizations that acquired software design technologies and supporting tools, they report that 70 % of modeling tools are never used after being available for one year, 25 % are used by only a single group, and only 5 % become widely used. Usability of modeling tools has been recognized as a possible factor limiting adoption (Agarwal et al. 2000) (Elaasar et al. 2017) (Agarwal & Sinha 2003) . Investigations of software engineering and computer science education programs suggest limited attention to software design concepts, accompanied by a general perception of lack of effectiveness of design notations such as UML (Badreldin et al. 2015)(Liebel et al. 2017). Recent studies of students' perception suggest that counter intuitively, students tend to view software design activities progressively less effective as they advance in a typical computer science or software engineering program (Iivari 1996).

Unfortunately, there is a lack of studies that investigate trends in the practices of software engineering as it relates to design and modeling activities. By understanding the trends over an extended period of time, researchers can focus their efforts on challenges that are the most relevant to practitioners and suggest most impactful features for modeling tools providers. This paper attempts to uncover the trends of adoption of software design and modeling in practice. Towards this goal, we designed a survey and collected data in three phases ten years apart. All the phases were conducted online and employed identical solicitations and questions. This paper extends previous work as follows (Badreddin et al. 2018). First, we designed and executed a post-survey study questionnaire and collected additional quantitative and qualitative data (Section 5.1), and report on the post-survey results (Section 5.2). Participants in the post-survey study were recruited from phase II. We also extend the reported data and extend the results and analysis of the study. This paper is organized as follows. In Section 2 we present related work covering related survey studies. In Section 3 we present the study methodology, effort to minimize bias and profiling summary data. In Section 4 we present the results of the survey for both phases. Section 5 presents the phase III post-study survey design and results. The data analysis identifying trends in the practice are presented in Section 6. We present threats to validity in Section 7. Finally, Discussion and Conclusion are presented in Section 8.

2. Related Work

Whittle et. al. conducted a survey of 450 Model Driven Engineering (MDE) practitioners to understand the extent to which practitioners adopt model-driven development styles (Whittle et al. 2014). Their survey results suggest that MDE adoption may be more widespread than commonly believed. They found that developers often do not use models to generate complete systems. Another survey conducted by de Silva aims at identifying key concepts and terminologies in the model driven engineering domain (Da Silva 2015a). Their goal is to provide answers for fundamental questions such as; what constitutes a model? what is the relationship between a model and a metamodel? and what are the fundamental facets of modeling languages? Another survey study of practitioners focused on practices in Turkey (Garousi et al. 2015). The authors report a significant level of UML use and design practices. They also found that the waterfall process, despite being old-fashioned, is broadly practiced.

In education, Badreddin et. al. collected 195 student responses from seven programs at four higher education institutions (Badreldin et al. 2015). The goal of the study is to uncover any trends in students' perceptions of the effectiveness of UML in software development. The authors found a consistent downward trend in how students perceive UML effectiveness as they progress towards their degree. In a similar study by Liebel that included 218 student responses, the authors found that UML modeling tool complexity to be a significant factor in education (Forward & Lethbridge 2008). They recommend the use of education-specific UML modeling tools to enhance students' learning experiences. They also found that if the industrial design and modeling tools were to be used in classrooms, a dedicated tool support resource becomes necessary. In another study that focused on embedded systems, the authors surveyed 275 engineers and found that model driven approaches enhance productivity and the portability of the developed embedded software (L. T. W. Agner et al. 2013). We summarize other survey studies of design and modeling practices in Table 1.

3. Methodology

A three-phase survey is conducted to understand the practice of software design and UML modeling which includes a post study questionnaire. The goal of this phase III post-study survey is to reflect on the phase I and phase II surveys key findings with participants and to collect additional data. We present the solicitation approach, survey structure and topics, efforts to minimize bias, and the demographics information of respondents. A complete report on the set of questions, methods, and specific measures taken to minimize threats to validity is published in a separately attached artifact as a technical report (Badreddin, Omar, Khandoker, Rahad, Forward, Andrew, Masmali, Omar, Lethbridge Timothy C. 2017) (Badreddin, Omar, Khandoker, Rahad, Forward, Andrew, Masmali, Omar, Lethbridge Timothy C. 2020).

	Scale/Region	Year	Paticipant Type	Count	Medium	Goal/focus area	Key Findings
(Anda et al. 2006a)	ABB	2016	Professionals	16	Online Interview	Assessment of UML based development	Evaluation of UML adoption in a specific project
(Forward & Lethbridge 2008)	Global	2007	Professionals	113	Online	Characterization of code-centric versus model-centric development methodologies	Identification of key benefits and challenges with model-centric development
(Garousi et al. 2015)	Turkey	2015	Professionals	202	Online	software engineering practices in Turkey	On average, design related activities consumed 12% of total project effort
(Badreldin et al. 2015)	U.S, Canada, Israel	2015	Students	195	Online	Trends in students' perceptions	Students graduate with overall negative perceptions of the effectiveness of UML
(Liebel et al. 2017)	U.S, Canada, Sweden	2017	Students	218	Paper	Case study in Model Driven Engineering Pedagogies	Tool complexity reduces MDE pedagogy effectiveness
(L. T. W. Agner et al. 2013)	Brazil	2013	Professionals	275	Online	Use of UML modeling and model-driven approaches for embedded software	Model-driven approaches provide productivity and portability of embedded software systems
(Dobing & Parsons 2008)	Global	2008	Professionals	284	Online	Different dimensions of UML usage	This study suggests several aspects of UML adoption and there is no standard approach to using UML within a group
(Badreddin et al. 2018)	Global	2018	Professionals	228	Online	Uncover software design and modeling practices	The results suggest some increase in formal and informal modeling and identify key challenges with modeling platforms and tools.
(Agarwal & Sinha 2003)	USA	2003	Students	N/A	Paper	An empirical study aimed at assessing the usability of UML	Developers gave very low score for UML usability. On the other hand, novice developers have positive perceptions of the usability of UML.
(Dobing & Parsons 2006)	Global	2006	Professionals	182	Online	Assess the perception of UML usage.	More extensive educational programs are needed, both to increase the number of analysts familiar with UML, and provide ongoing support to help them make fuller use of its capabilities.
(Cherubini et al. 2007)	Microsoft	2007	Professional	350	online	To understand the perception of using whiteboard drawing by the developers	One of the major findings of this study is that developers use informal notation to support face-to-face communication and existing tools do not have support to externalize their mental models of code.
(LaToza et al. 2006)	Microsoft	2006	Professional	1000	online	To understand developers' typical tools, activities, and practices and their satisfaction level	The study finds that face-to-face communication is beneficial and developers spend a significant amount amount of time recalling their mental model.

Table 1 Summary of survey studies of design and modeling practices

3.1. Solicitation

The survey was conducted online (Rahad Khandoker and Omar Badreddin 2017). We sent targeted solicitations to a wide variety of organizations and posted the survey on many forums, including Javaranch (Programming Forum 2017c), Javaforum (Programming Forum 2017b), Dream in code (Programming Forum 2017a), several UML user groups, and agile methodology user groups. We posted the survey on technological websites, such as digg.com and dzone.com. We also posted the survey in different Facebook programming groups such as JavaScript, Php Programming, and others. We identified these Facebook programming group's popularity in phase II by the total number of members and ensured that these groups are active. The venues such as digg.com, dzone.com, Javaforum, Javaranch, UML user groups were popular in both phases. We followed identical solicitation venues, techniques, and frequencies for both phases of the study.

3.2. Survey Structure

The survey consisted of nine topics. Each topic is explored by a set of questions with possible answers varying from a 5-point Likert scale, to open-ended free text questions. The Likert scale range is from SD (Strongly Disagree) to SA (Strongly Agree). The abbreviations of 'D' and 'A' are respectively Disagree and Agree. Some questions included additional options of Never, Always, and Not Applicable. Only complete responses were included in the analysis, but participants were able to skip some questions based on their answers. For example, participants who did not have experience with a specific approach were able to skip all related sub-questions. In total, the survey included 18 questions. The survey is expected to take about 60 minutes for completion. The survey topics are as follows.

Topic 1 (fundamentals): This topic constructs a characterization on what constitutes a software design activity and a software model. Various options were presented ranging from class diagrams, use cases, to source code. The objective is to uncover any perceived notions of what constitutes a design or modeling activity.

Topic 2 (basic characterization of the practices): This topic explores the basic characteristics of the practice, including how the modeling and designing activity is accomplished and when, and what notations are used in the course of software design. The objective of this topic is to develop a basic understanding of the state of the practice.

Topic 3 (life cycle): This topic investigates the activities involved across various development phases from requirements, design, testing, and documentation. The goal is to characterize the various activities performed throughout the entire software life cycle.

Topic 4 (platform): This topic explores the specific tools, methods, technologies, and platforms used in the software development activities. This topic also documents the characteristics of the nature of the software applications that participants develop.

Topic 5 (efficacy): This topic investigates specifics about the design and modeling practices. The questions under this topic explore the various activities and inquire about the efficacy of the various approaches for the task at hand.

Topic 6 (code versus model centrism): This topic explores perceived challenges in code-centric software development approaches and perceived challenges in model-centric development approaches.

Topic 7 (open-ended and optional contact information): This topic includes an open ended set of questions and comments about the software design practices and questions about the survey itself. This includes optional questions where the participants can voluntarily provide contact information for follow up.

Topic 8 (demographics): Demographics question with sub-questions that include country of residency, education level, and years of experience of the participant.

3.3. Minimization of Bias

To minimize bias, we employed two survey techniques; randomization of questions and balancing positive/negative questions. Randomization was applied to the order of questions to ensure that possible participant fatigue is distributed across survey questions. Specifically, the following question order was randomized: question 2 to question 5 inclusive, and question 7 to question 17 inclusive. To minimize bias in question wording, we adopted neutral question wording whenever possible. In cases where neutral wording was not possible, we balanced the number of positive and negative phrases. For example, questions that explore the advantages of a specific design and modeling activity are balanced by questions about the advantages of code-centric approaches. Moreover, in multiple steps in the survey, we stated additional information so that participants could consistently answer a set of questions, as an example:

"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 whiteboard, 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." Since the goal of this study is to discover trends, any biases inherited in the survey structure and questions will be present in three phases, and therefore will be largely minimized in the course of trends analysis.

3.4. Demographics

The survey collected demographics information related to years of experience, the level of education of participants, their geographical location, and the nature of their professional software engineering role and activities they perform, and the nature of application they develop. Sub-sample analysis is conducted and is reported in the included artifacts and technical report. In Table 2, we present the demographics summary data for phase I and phase II separately.

4. Results

We present the results of the survey as follows. All questions under the same topic are combined within the same table whenever possible. For brevity, we only show the combined values for Strongly Agree and Agree, and combined values for Strongly Disagree and Disagree. We also list the mean value for phase I and phase II separately. Mean values are calculated by converting the Likert scale to a vector of values from 5 corresponding to Strongly Agree, to 1 corresponding to Strongly Disagree. For each topic, we also show the mean gap, calculated by the difference between the mean value for phase I and phase II. A significant mean gap is shown in bolded and underlined font in each table. The null hypothesis is that the two values of phase I and phase II come from the same sample. We made an assumption that when two snapshots mean difference is 0.4 and above, we consider this a significant difference. A significant difference would indicate divergence (either upward or downward) in the survey data.

4.1. Topic 1: Fundamentals

The primary objective of questions in this topic is to ensure that participants have a general agreement on what constitutes a model. This topic included questions about class diagrams, deployment diagrams, picture by a drawing tool, textual use case, whiteboard drawing, picture by hand, source code, source code comments, and others.

The results of this topic are shown in Table 3. The perception of modeling is slightly changed downward recently in terms of picture by drawing tool, textual Use Case, whiteboard drawing, and picture by hand (Table 3).

4.2. Topic 2: Characterization of Practices

This topic focuses on uncovering how participants perform their modeling and design activities, and how they learn about various aspects related to modeling. Participants are asked about the methods they use to create models and are provided with choices that include whiteboard drawing, diagramming tool, word processor, word of mouth, hand written material, comments in source code, modeling or CASE tool, drawing software, and others. On how participants learn about modeling, the questions focused on the type of artifacts they referred to and how they go about their learning activities.

The results for this topic is shown in Table 4 and 5. Recently, "word of mouth" is less used by the practitioners whereas comments in source code, modeling tools, and drawing software are

Category	Phase I		Phase II	
	N	%	N	%
All participants (complete responses only)	113	100	115	100
Participants in U.S/Canada	63	55.7	47	41
Participants in EU	13	11.5	3	2.6
Participants outside US/Canada and EU	14	12.5	16	13.8
Participants with >12 years of experience	60	53.47	55	46.9
Participants with PhD	9	8	12	12.1
Participants with Masters	40	35.4	34	36.4
Participants with Bachelor	35	31	12	12.1
Participants without degrees	29	25	23	19.1

Table 2 Demographic data

Responses for Topic 1: What is a software model							
Entity that might be a model	**Phase I**			**Phase II**			**Mean Gap**
	% SA+A	**% SD+D**	**Mean**	**%SA+A**	**%SD+D**	**Mean**	
Class Diagram	88.4	2.7	4.3	87	4.9	4	-0.3
UML Deployment Diagram	77.5	5.4	4.1	72	17.5	3.8	-0.2
Use Case Diagram	82.1	9.8	4	80	13.5	3.8	-0.3
Picture By Drawing Tool	**85.6**	**7.2**	**4**	**62**	**20.3**	**3.5**	**-0.5**
Textual Use Case	**78.8**	**10.6**	**4**	**59**	**18.4**	**3.5**	**-0.5**
Whiteboard Drawing	**78.8**	**8.8**	**3.9**	**63**	**20**	**3.6**	**-0.4**
Picture By Hand	**57.1**	**9.8**	**3.9**	**61**	**13.4**	**3.5**	**-0.4**
Source Code	46.8	38.7	3.2	47	38.7	3.1	-0.1
Source Code Comment	33.9	41.1	2.9	44	39.9	3	0.1

Table 3 What is a software model?

frequently used as a medium and methods of modeling (Table 4). Software practitioners are more inclined to use models for brainstorming possible designs. However, there is a decline in using models as a prototype (Table 5).

Another set of questions are related to the type of artifacts the developers refer to as shown in Table 6. Table 6 depicts a contradictory result between phase I and phase II data on using word processor and diagramming tool as a reference.

The last set of questions in this topic explores participant's daily activities as shown in Table 7. This result shows a positive perception of using modeling in daily activities such as fixing bugs and write/maintain test scripts in phase II.

4.3. Topic 3: Lifecycle

Questions in this topic focus on modeling and design activities as it relates to the project lifecycle. The first set of questions explore when design activities take place in relation to coding. The choices for this question include before coding, during coding, after coding, and only on request.

The second set of questions in this topic explores the various activities that are performed by the participant. These activities included searching, requirements, design, modeling, testing, coding, transfer, develop tests, and documentation.

The third set of questions investigates the daily activities of participants. The survey provided choices that participants could choose from and included additional fields where participants could identify additional activities. The choices that were provided include think about a software system, run or attend meetings, explain software design to others, design a software system, lead software project, conduct a search about a software system, model a software system, write new code, maintain existing code, fix bugs, perform manual testing, write or maintain requirements, general administration, and write or maintain test scripts. The data for this question is used primarily in sub-sample analysis.

Software modeling and various activities during project life cycle are shown in Table 8 and 9. Table 8 data shows that activities in the project life cycle such as searching design testing, knowledge transfer, documentation are less performed in phase III compared to phase II. Table 9 significantly shows that models are only created on request supported by phase III data.

4.4. Topic 4: Platforms

For the platform topic, we explore the modeling notations that participants use. Some choices that were provided to participants in the survey included UML (any version), UML 2.*, SQL, Structured Design models, UML 1.*, ERD, Well-defined DSL (DSL that has concrete unambiguous semantics), ROOM / RT for UML, SDL, Formal (e.g. Z, OCL), BPEL, and others. This topic also explores the use of various modeling tools, such as Eclipse, Visual Studio, Rational RSx, Rational XDE, and others. The topic also explores various related technologies and platforms, such as Java, PHP / Perl, ASP.Net, Ruby / Python, C / C++*, and others.

Summary results of different platform usages are shown in Tables 10, 11 and 12. Table 10 data shows that older version of UML usages declined among practitioners. However, there is a significant increase in using domain-specific languages and formal modeling such as OCL in phase II. Table 11 provides contradictory results on Eclipse usages as a development tool. The result shows that the Eclipse platform usages decreased over time. Table 10 data depicts that Java programming language was used often in phase I whereas Ruby/Python is being used more frequently (phase II data).

Topic 2: Medium and methods of modeling							
Medium or methods used to model	**Phase I**			**Phase II**			**Mean Gap**
	% Never&Sometimes	**% Very Often**	**Mean**	**% Never& Sometimes**	**% Very Often**	**Mean**	
Whiteboard drawing	33.3	45.0	3.2	40	57.9	2.9	-0.3
Diagramming tool (e.g. Visio)	42.3	36.9	2.9	43	43.2	2.8	-0.1
Word processor / text	45.5	26.8	2.8	42	55.3	2.7	-0.1
Word of mouth	**42.3**	**27.0**	**2.8**	**54**	**46.1**	**2.4**	**-0.4**
Handwritten material	51.4	22.5	2.6	49	51.3	2.6	0.0
Comments in source code	51.4	21.6	2.5	49	37.8	2.6	0.1
Modeling tool/CASE	58.9	29.5	2.4	55	29.0	2.5	0.1
Drawing software	72.1	12.6	2.1	68	29.0	2.3	0.2

Table 4 Medium and methods of modeling by survey participants

4.5. Topic 5: Software Development Domain

This topic presents the software that participants develop in their profession. The participant selected one of the following options for each sub-question listed: Never, Sometimes, Moderately often, Very often, and Always. We combined Never and Sometimes together and Often and Very often together and presented the results in Table 13.

The software built by practitioners and software developers includes Computational-dominant software (e.g., Simulation, Scientific, Image Processing, Machine Learning), Business software (e.g., Bank Transaction Processing, Financial Analysis, GIS, Software Tools), Consumer software (e.g., Word Processors, Spreadsheets, Browsers, Games), Information display and transaction entry (e.g., Search Engines, Maps, Weather, News), Operating systems (e.g., Mac, Windows, Linux), Middleware and system components (e.g., Database servers, Virtual Machines), System Support utilities (e.g., Security, Anti-Virus, Spam Filter, Encryption), Website content management, Servers (e.g., Email, IM, Proxies, Load Balancers), Malware (e.g. Virus, Spyware, Spam), Embedded real-time software (e.g., Firmware, Routers), Industrial control software (e.g., Air Traffic Control), Design and engineering software (e.g., Testing tools, Development environments, Database / Reporting, Modeling Tools) etc.

Summary results of Table 13 regarding the type of softwares that are built by practitioners show that most often is related to business (phase II data).

4.6. Topic 6: Efficacy

This topic presents questions related to the suitability of the modeling tools for the targeted activities, i.e. developing a design, transcribing a design into digital format, generating code where code is accessible and editable, prototyping a design, brainstorming possible designs, generating all code (no manual coding), and others (Table 15). The topic also explores participants' perceptions of key characteristics of modeling tools (i.e. modeling tools suitability as a medium of communication with other developers, ease and speed to create models, suitability for model-based analysis, support for collaborations, visualization of different aspects of the models, generate code, embed parts of models in documents, etc.) shown in Table 14.

There is a positive mean gap for generating all the code from the model between phase I and phase II which implies that modeling tools are suitable at this task in recent years (Table 15). However, the performance of modeling tools at developing a design is decreased in phase II. Similarly, a positive mean gap is visible for the ability to view different aspects of a model and information density in Table 14.

4.7. Topic 7: Code Versus Model Centralism

This topic investigates in-depth perceptions of participants on when coding approaches versus modeling and design ap-

Topic 2: What models are used for?							
Activity	**Phase I**			**Phase II**			**Mean Gap**
	% Never & Sometimes	**% Very Often**	**Mean**	**% Never & Sometimes**	**% Very Often**	**Mean**	
Developing a design	26.6	48.4	3.3	28	55.1	3.2	-0.1
Transcribing a design into digital format	32.8	39.1	3.1	41	51.7	2.9	-0.2
Prototyping a design	**53.1**	**32.8**	**2.7**	**24**	**32.2**	**2.2**	**-0.5**
Brainstorming possible designs	**54.7**	**23.4**	**2.6**	**34**	**44.8**	**3**	**0.4**
Generating code (code editable)	65.1	17.5	2.2	66	34.4	2.2	0
Generating all code	76.6	14.1	1.8	66	31	2.1	0.3

Table 5 What models are used for

Responses for Topic 2: Reference materials							
Refer to material created by/as	Phase I			Phase II			Mean Gap
	% Never and Sometimes	% Very Often	Mean	% Never and sometimes	% Very Often	Mean	
Word of mouth	22.3	54.5	3.4	40	60.5	3.1	-0.3
Word processor / text	**30**	**48.2**	**3.3**	**29**	**54**	**2.9**	**-0.4**
Diagramming tool	**32.4**	**42.3**	**3.1**	**70**	**36.9**	**2.7**	**-0.4**
Whiteboard drawing	34.5	41.8	3	37	48.6	2.7	-0.3
Comments in source code	42	30.4	2.9	55	47.3	2.7	-0.2
Drawing software	57.8	13.8	2.6	32	39.5	2.4	-0.2
Modeling tool/CASE	55.9	31.5	2.5	85	28.9	2.3	-0.2
Handwritten material	56	20.2	2.4	27	29.7	2.3	-0.1

Table 6 Reference materials

proaches are more suitable to perform various activities. These activities include fixing a bug, creating efficient software, creating a system as quickly as possible, creating a prototype, creating a usable system for end-users, modifying a system when requirements change, creating a system that most accurately meets requirements, creating a re-usable system, creating a new system, comprehending a system's behavior, explaining a system to others etc. The answer choices for these questions ranges from much easier in models, to much easier in code. The summary results of this topic are shown in Table 16.

Table 16 results show a positive mean gap between phase I and phase II data in creating reusable systems. This implies respondents perception regarding creating reusable system remains easier in code centric approach according to the phase II survey study.

Table 17 and Table 18 results depict problems with model and code centric approach respectively. Table 17 reports that there is a positive mean gap regarding model and code consistency between phase I and phase II data. This refers that models and code consistency is a major concern for software practitioners recently. Additionally, programming languages likely to become obsolete is a concern in phase II (Table 18).

4.8. Open ended and follow up questions

This topic included multiple open-ended questions about modeling and coding, questions about the survey, and optional contact information for follow up questions. Those who provided their contact information from the phase II survey were contacted with summaries of the survey results and were provided the option for additional feedback on the survey itself and on the

Responses for Topic 2: Daily activities of participants							
Available tasks	Phase I			Phase II			Mean Gap
	% Never&Sometimes	% Very Often	Mean	% Never& Sometimes	% Very Often	Mean	
Think about s/w system	9.4	77.1	4.1	12	41.2	4.1	0
Run / attend meetings	19.8	60.4	3.6	14	68.6	3.5	-0.1
Explain s/w design to others	15.8	51.6	3.5	26	65.7	3.2	-0.3
Design a s/w system	18.8	57.3	3.5	34	54.3	3.3	-0.2
Lead software project	29.2	53.1	3.3	23	65.7	3.2	-0.1
Search about s/w system	31.2	46.2	3.2	31	51.4	3.2	0
Model a s/w system	30.2	45.8	3.2	37	45.8	3.1	-0.1
Write new code	37.5	49	3.1	29	54.3	3.3	0.1
Maintain existing code	37.5	40.6	3	26	60	3.3	0.3
Fix bugs	**39.4**	**39.4**	**3**	**23**	**48.6**	**3.5**	**0.5**
Perform manual testing	35.1	34	2.9	37	51.4	3.1	0.2
Write / maintain requirements	41.1	40	2.9	34	48.6	3.1	0.2
General administration	40.4	29.8	2.8	43	54.3	2.8	0
Write / maintain test scripts	**58.3**	**17.7**	**2.4**	**47**	**44.1**	**2.8**	**0.4**

Table 7 Daily activities of participants

Topic 3: When do you perform the following tasks?					
Available tasks	**Phase I**		**Phase II**		**% Gap**
	Mode	**%**	**Mode**	**%**	
Searching	Constantly	64.5	Constantly	36.1	-28.4
Requirements	Start	60	Start	72.2	12
Design	Start	53.8	Start	44.4	-9.4
Modeling	Start	46.5	Start	66.7	20.2
Perform testing	Constantly	44.1	Constantly	42.9	-1.2
Coding	Constantly	41.7	Constantly	31.4	-10.3
Knowledge transfer	Constantly	41.7	Constantly	30.6	-11.1
Develop tests	Constantly	40.2	Constantly	34.3	-5.9
Documentation	End	38.7	End	27.8	-10.9

Table 8 Various activities throughout the project lifecycle

results. The following are itemized summaries of analysis of the text collected in open-ended and follow up questions for both survey phases (phase I and II). We removed the repetitive information from the original free text without changing the main content of the respondents' response. We adopted a similar coding technique from the grounded theory (Anda et al. 2006b) in the process of analyzing the qualitative data. When we reached theoretical saturation: the point where we have sampled and analyzed our data until we have uncovered all the data. After reaching out the saturation point we ignored repetitive texts in qualitative data.

Adopted Technique:

1. The participants often expressed themselves in many words, so some of the sentences from the participants were simplified to facilitate the rest of the analysis.

2. All sentences related to possible software modeling and deigning practices were sorted into positive and negative perspectives.

3. Participants' background is checked by identifying their years of experience in software engineering and software modeling.

– *I have taken courses on UML and software design, but the "culture" here has not yet adopted these concepts. They try to use UML, but merely for analysis, not development. We do not yet have case tools or modeling tools.*
– *In "the real world" it's necessary to cut these models down to the bare basics. Adding too much details 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 should be used to validate and share your design ideas. If your model works, you can build it too; and others can learn it more easily. Anything more is a waste of time, anything less will cost you more time in the long run.*
– *There are a time and cost associated with producing the model upfront, but that time is more than gained back during development and, especially, maintenance.*
– *Now if the [modeling] tools were actually developed by modelers they'd be much better! The tools are often too code centric.*
– *Code wins over models every time when it comes to revenue. Working and tested code with business value can be sold. Models don't sell, well, unless you are in a huge defense contracting world.*
– *Model based systems tend to be far more reliable.*
– *One day there will be no need for learning languages to develop systems.*
– *Modeling using documentation / plain English is great. Modeling using any formal language or tool has been proven useless for all of my applications.*
– *First, we have to model the software according to the requirements fulfilling of end-user [needs/requirements].*
– *I love modeling but our team does not have a culture of doing it and it is primary because they do not think we have enough time. Inevitably we waste lots of time because of our lack of models.*
– *I've been a professional in the industry since 1981. CASE tools were popular and successful in limited domains, but are hard to maintain.*
– *I have done a fair bit of software modeling but it's been informal. I have no experience with design programs and only academic experience with things like UML.*
– *Modeling and coding both should work together.*

5. Post-survey study

Following the survey, we conducted a post-survey (phase III) study with a subset of survey participants (Rahad Khandoker and Omar Badreddin 2018) from phase II. The study included eighteen open-ended questions under three categories; modeling tools and platforms, the practices of modeling, and follow-up and profiling questions. Each question included a Likert-scale and an open-ended component to solicit qualitative data. In this section, we present the post-survey design and the results.

5.1. Post-Survey Study Design

The goal of this post-study questionnaire is to reflect on the surveys (phase I and phase II) key findings with participants and to collect additional quantitative and qualitative data. As such, this study was designed after the survey data was collected and analyzed.

We solicited participation from the pool of survey participants who provided their contact information and indicated a willingness to participate in the follow-up study from phase II. Twenty participants' responses are included in the study. In each solicitation, we presented the participant with a brief summary of the key study results and findings.

5.2. Post-Survey Study Results

We contacted 150 software practitioners for the post-survey study from phase II. We received twenty complete responses.

Topic 3: When is modeling performed?							
Timeline	**Phase I**			**Phase II**			**Mean Gap**
	% Never&Sometimes	**% Very Often**	**Mean**	**%Never & Sometimes**	**% Very Often**	**Mean**	
Before coding	18.8	59.8	3.7	16	54	3.7	0
During coding	33.3	36	3.1	41	51.3	2.8	-0.3
After coding	60.4	19.8	2.5	54	37.8	2.5	0
Only on request	**78.5**	**10.3**	**1.9**	**59**	**32.4**	**2.3**	**0.4**

Table 9 When is modeling performed?

More than half of post-survey study responses (56%) reported their role as a software designer, and 20% software developers and 25% reported their role as an educator. The following Table 19 summarizes how much software design and modeling those participants have performed in the last year. In the following, we present the phase III post-study quantitative and qualitative data.

5.2.1. Quantitative Data The most prominent result in the post-study survey is the concerns about future support for both modeling tools and development platforms. More than 80% reported being strongly concerned that modeling tools may not be supported in the future in Table 20. This is related to significant concerns regarding the interoperability of modeling tools and the reusability of their models. More than half of the participants reported having increased their use of Domain-Specific Languages and code generation for all or parts of the software system (Table 20). This is consistent with the findings of phase I and II survey data.

The post-study data did not indicate that modeling tools are complex, difficult to learn, or difficult to use productively in a software development project. For example, 93% did not find modeling tools difficult to learn (Table 20). We attribute this unexpected result to the fact that those who opted to participate in the post-study survey are more experienced heavy users of advanced modeling tools. This is evident by the fact that a significant portion of those participants reported using formal modeling and Domain-Specific Languages in their development activities. Phase III post-survey study results are summarized in Table 20 and 19.

5.2.2. Qualitative Data Each question included an open-ended section where the participants were able to provide additional free-text responses to the questions. The responses are taken from the phase II respondents who participated in post survey study (phase III).

Responses reflect broad and diverse experiences with software design and modeling, from very informal and casual modeling practices to formal and DSL-based experiences. A recurrent theme is the lack of support from senior developers and managers for acquiring and supporting software design tools. The lack of commitment from senior developers and management is that modeling incurs significant upfront costs associated with technology and tools acquisition and personnel training.

Topic 4: Modeling notations and tools							
Modeling notations	**Phase I**			**Phase II**			**Mean Gap**
	% Never&Sometimes	**% Very Often**	**Mean**	**% Never & Sometimes**	**% Very Often**	**Mean**	
UML (any version)	30.9	51.8	3.3	46	33.4	2.9	-0.4
UML 2.*	52.1	34.4	2.6	53	34.4	2.5	-0.1
SQL	55.6	29.6	2.5	49	34.3	2.7	0.2
Structured Design models	58.8	21.6	2.5	50	38.2	2.7	0.2
UML 1.*	**54.8**	**28**	**2.4**	**73**	**26.7**	**1.9**	**-0.5**
ERD	**63.2**	**20.8**	**2.3**	**46**	**40**	**2.9**	**0.6**
Well-defined DSL	**78.8**	**5.8**	**1.7**	**62**	**32.3**	**2.4**	**0.7**
ROOM / RT for UML	85.9	7.1	1.5	79	15.2	1.8	0.3
SDL	**89.2**	**3.2**	**1.3**	**68**	**25.8**	**2.2**	**0.9**
Formal (e.g. Z, OCL)	**93.9**	**2**	**1.3**	**75**	**18.8**	**1.9**	**0.6**
BPEL	92.8	3.1	1.3	87	13	1.6	0.3

Table 10 Modeling notations and tools

Topic: Development Tools	Phase I			Phase II			
Available options	Never and Sometimes	Very Often	Mean	Never and Sometimes	Very Often	Mean	Mean Gap
Eclipse	**43.9**	**40.8**	**3.0**	**71**	**20.0**	**2.1**	**-0.9**
Visual Studio	56.7	32.0	2.4	63	25.7	2.3	-0.1
Rational Rose	76.5	17.3	1.8	85	14.7	1.5	-0.3
Rational RSx	85.7	10.2	1.4	94	5.8	1.2	-0.2
Rational XDE	89.7	5.2	1.4	94	5.7	1.2	-0.2

Table 11 Development tools

Technology options	Phase I			Phase II			Mean Gap
	% Never& Sometimes	% Very Often	Mean	%Never & Sometimes	% Very Often	Mean	
Java	**46.3**	**31.6**	**2.4**	**80**	**11.5**	**1.8**	**-0.6**
PHP / Perl	74.2	19.4	2	74	14.3	2.2	0.2
ASP.Net	79.4	14.4	1.8	74	14.3	2	0.2
Ruby / Python	**88.3**	**8.5**	**1.6**	**77**	**17.2**	**1.9**	**0.3**
C / C++*	60	30	2.4	65	25	2.3	-0.1

Table 12 Technology

These upfront costs are difficult to justify particularly that the pay-off typically occurs later in the development cycles during maintenance and post-deployment.

Interestingly, developers who work on long-living software code bases tend to be strongly in favor of modeling and design. With those developers, the primary concern has been the sustainability of the technology in face of tool and platform changes. We also observed significant interest in modeling to aid in documentation and communication. In these cases, casual informal modeling tools were favored.

Another recurrent theme, pertaining to modeling tool usability, is the need for tool customization. We infer that those customizations are beyond what can be achieved by end-users or by manipulating basic tool options. This emerges as a reason for the lack of adoption of technology in several cases.

The following are itemized summaries of analysis of the text collected in open-ended questions in phase III. We removed the repetitive information from the original free text without changing the main content of the respondents response. The phase III free text analysis technique is similar to the phase II approach which is discussed in detail in section 4.8.

– *We use a modeling tool that we developed in-house. Our DSL generates Scala that I never inspect. In fact, I do not really know Scala. That is why I am not concerned about the code generation or synchronization between the model and the code. But we have test cases against the generated code.*
– *I tend to develop rather simple models. The modeling tools may be difficult to learn if I were to develop more complex models with different notations. But for my needs, the modeling tool we use [Papyrus] is not difficult to learn. Note that documentations are scarce and frequently outdated. This will make it difficult for new users.*
– *I use modeling tools primarily to share ideas and design with my colleagues. Even though our code is open source, we almost never share models in the open-source. If we do, how can [other developers] use it or edit the model? Also, the model is usually different than the implementation, so we do not want to confuse those who want to commit code. We could share models as an image, but then it is static and will become obsolete fairly soon. I guess from that perspective, interoperability with other modeling tools is important.*
– *We would use much more modeling and invest more in tools and training if we can have a business case. How can we show that it is [design activities and modeling] useful? It is only useful when maintenance becomes significant and that usually happens many years after development.*
– *I deal with long term projects and we know [the value or usefulness] of these tools. We recently changed all the modeling tools we use, who knows what will happen next. Marketing of design tools is weak and it is difficult to convince others.*
– *As a consultant, I see how companies always change their programming languages. [Programming languages] always change. With models it is much more stable [because] you can always recreate the model in another tool. It takes some time but that is much less than code.*
– *Models are generally a product of the design process which includes brainstorming activities.*

Topic: What types of software do you build?	**Phase I**			**Phase II**			
Available options	**% Never & Sometimes**	**% Very often**	**Mean**	**%Never&Sometimes**	**%Very Often**	**Mean**	**Mean Gap**
Business	**44.8**	**45.8**	**2.9**	**30.4**	**75.1**	**4.12**	**1.22**
Design and Engineering	60.4	25	2.4	50.23	46.2	2.57	0.17
Website Content Management	62.1	23.2	2.3	67.2	27.9	2.11	-0.19
Information Display (Search / News)	66	26.8	2.2	64.3	36.1	2.41	0.21
Middleware	67	23.7	2.2	70.2	38.1	2.44	0.24
Consumer	67.7	21.9	2.1	60	28.1	2.07	-0.04
Operating Systems	**74.00**	**21.90**	**2.00**	**69.30**	**26.80**	**2.48**	**0.48**
Computational	**76.6**	**11.7**	**1.9**	**80.7**	**33.7**	**2.37**	**0.47**
Servers	**75.3**	**12.4**	**1.9**	**60.4**	**17.9**	**1.49**	**-0.41**
Embedded Real-Time	76.8	14.7	1.8	104.7	16	2.1	0.3
System Utilities	84.2	7.4	1.6	80.2	21.4	1.69	0.09
Industrial Control	89.5	9.5	1.5	79.5	16.9	1.59	0.09
Malware	**92.7**	**2.1**	**1.2**	**103.7**	**17.9**	**1.83**	**0.63**

Table 13 Software development

- *The modeling tool I use is very easy to learn.*
- *There are models that we could not create without customization of the tool. We had to change the tool and customize boxes and elements.*
- *The benefits of modeling is long term.*
- *Models are the best way to communicate with clients. But for tools, some tools are easy some are not.*
- *Code generated from models is reusable.*
- *Modeling tools are most effective for requirements elicitation and analysis.*
- *I just started with MDE a few years ago.*
- *I have not increased my use of DSLs (in the last few years), I have been using them quite a lot.*
- *I love coding. I feel I will not start using code generation tools.*
- *It has become easier to use code generation [recently].*
- *We did use whiteboards it was more practical for brainstorming. Once we had the idea we used a digital modeling tool.*
- *(synchronization between the model and the code) has become better for sure, but it is still a concern. [I am] giving up.*
- *Sometimes modeling if not done right can be different than the code. As we code new solutions to problems can emerge and the model might change.*

What are the desired attributes of a modeling tool?							
Modeling tools attributes	**Phase I**			**Phase II**			**Mean Gap**
	Rank	**Mean**	**SD**	**Rank**	**Mean**	**SD**	
Communicate to others	**1**	**5**	**4**	**1**	**4.2**	**4**	**-0.8**
Readability	**2**	**5**	**4**	**2**	**4.5**	**3**	**-0.5**
Ease and speed to create	3	3	2	3	3.3	3	0.3
Ability to analyze	4	3	3	4	3.1	3	0.1
Collaborate amongst developers	5	3	2	5	4	3	1
Ability to view different aspects of a model	**6**	**2.3**	**2**	**6**	**3**	**3**	**0.7**
Generate code	7	3	2	7	3	2	0
Information density	**8**	**3.2**	**2**	**8**	**3.9**	**3**	**0.7**
Embed parts of model in documentation	9	3	2	9	2.8	2	-0.2

Table 14 Modeling tools

6. Analysis

We provide the analysis of the results as follows. We present the upward and downward trends as exhibited in the survey results over the ten-year period. We also present what can be considered as a positive trend and a negative trend. We then summarize the expected and unexpected results. Finally, we present persistent challenges and emerging opportunities.

6.1. Upward and Downward Trends

We present the analysis of the data based on upward and downward trends as it is manifested in the change between the data set for phase I and phase II.

Upward Trends. There is a significant uptake in the use of well-defined and formal modeling languages, such as OCL, as well as well-formed Domain-Specific Languages (DSLs) [Well-defined and well-formed DSL refers that DSL has concrete unambiguous semantics]. This is also consistent with significantly more participants reporting generating all system code automatically from models (forward engineering), as well as more participants reporting not editing the generated code. This claim also reflects by participants from phase III data.

In cases where all code is automatically generated, there is less of a concern about the synchronization between the models and code as the system continues to evolve. This is evident in a significant decrease in participants' concerns about model and code becoming out of synchronization. This is also supported by the sub-sample analysis showing a high correlation between participants who reported generating all or most of the code and their responses showing little or no concern about code/model synchronization. This claim is consistent with phase III survey data where practitioners agreed that code generation and synchronization became easier recently. However, He et al. found that maintaining the synchronization between models and code is a challenging task during evolving Model-Based projects (He et al. 2016). They further reported that because of the poor support for model-code synchronization, when a piece of generated code is modified, it becomes inconsistent with the model. This makes the modified generated code can only be maintained manually by developers. This is an indication of perception change regarding synchronization between model and code among software practitioners. The interpretation is that now, software practitioners are more inclined to use the MDE approach for code generation because of refined modeling tools.

The creation of complete fully-executable models often involves a number of modeling notations and languages and requires the specification of multiple aspects of the system under design and development. This is reflected by the results in Table 14 where significantly more participants reported that it is very important that the modeling tool be able to view multiple aspects of the model under development.

Another significant, and related, upward trend is participants reporting that one of their most desired features in modeling tools is an ability to provide a high level of information density (Table 14). This phenomenon is also supported by Kleppe et al. (Kleppe et al. 2003) where the authors explained abstraction is the primary use of models. Further, the authors discussed that the creation of machine-readable, highly abstract models that are developed independently of the implementation technology and stored in standardized repositories. Our survey results are also supported by recent study (Khelladi et al. 2020) that discusses software modeling which aims to tackle increasing software development complexity by using abstraction. Abstraction is achieved by creating and reasoning on various models expressing various concerns of a software system, e.g., structural, behavioral, requirements, and business aspects.

There is also evidence for the significant increase in the use of data modeling using ERD models (Table 10). There are more participants who reported creating models 'only' upon request.

We are living through a significant flux in middleware, platforms, and technologies. This fact seems to be reflected in two ways in Table 17; 1) a significant increase in participants concern about tool providers not continuing to provide support for their modeling tools, and 2) a significant increase in concerns related to programming languages and related technologies become quickly obsolete. The modeling tools learning curve is also high which is iterated several times in the post-survey study qualitative data. Similarly, Mohagheghi et al. found that

How good are modeling tools for?							
Available activities	**Phase I**			**Phase II**			**Mean Gap**
	% Poor	**% Good**	**Mean**	**% Poor**	**% Good**	**Mean**	
Developing a design	**16.9**	**47.9**	**3.4**	**11**	**53.6**	**2.9**	**-0.5**
Transcribing a design into digital format	24.6	42	3.2	25	60.7	3.3	0.1
Generating code (code is editable)	39.1	29	2.9	32	64.3	3	0.1
Prototyping a design	41.2	29.4	2.9	25	71.4	3.1	0.2
Brainstorming possible designs	45.1	32.4	2.8	18	74.7	3.1	0.3
Generating all code (no manual coding)	**79.7**	**8.7**	**1.9**	**50**	**42.9**	**2.5**	**0.6**

Table 15 Modeling tools good at

Topic 6: Available activities	Phase I			Phase II			Mean Gap
	% Easier in Models	% Easier in Code	Mean	% Easier in Models	% Easier in Code	Mean	
Fixing a bug	28.9	43.3	3.2	19	40.6	3.2	0
Creating efficient software	35.9	43.5	3.1	27	50	3.2	0.1
Creating a system as quickly as possible	46.7	42.4	3	31	56.2	3.2	0.2
Creating a prototype	43	32.6	2.9	44	37.5	2.7	-0.2
Creating a usable system for end users	42.4	22.8	2.7	49	27.3	2.4	-0.3
Modifying a system when requirements change	54.9	24.2	2.5	41	37.5	2.8	0.3
Creating a system that most accurately meets requirements	67	19.8	2.2	56	26.4	2.3	0.1
Creating a re-usable system	**63**	**15.2**	**2.2**	**42**	**30.4**	**2.6**	**0.4**
Creating a new system overall	68.5	20.7	2.2	64	24.2	2.3	0.1
Comprehending a system's behaviour	71.9	15.7	2	75	15.7	1.9	-0.1
Explaining a system to others	81.8	7.6	1.7	66	15.6	1.9	0.2

Table 16 Tasks are better in a model centric versus code centric approach

user-friendliness of modeling tools and the provision of features for managing models of complex systems are crucial for wider industrial adoption of modeling tools (Mohagheghi et al. 2013). Companies have not yet started to apply model-driven approaches due to the associated cost and risks (heavy changes to the software development process are required), the lack of expertise, "immature" tools, or the lack of insight into the contexts in which the approach can give useful results. Additionally, the authors argued that tools must be improved regarding usability, multi-user support, versioning of models, and diff/merge possibilities.

Participants reported a significant increase in the need for creating reusable designs and systems and a significant increase in using models in brainstorming sessions.

There also seems to be a broadening in the methods to approach modeling as evident in increase in casual modeling using pen and paper which is consistent across all three phases. This is accompanied by participants reporting generally a stricter definition of what constitutes a model (i.e. more participants view that a textual use case is not considered a model).

There is also a positive trend of more participants viewing the generated code as being suitable for their end purposes and its quality matches or exceeds their expectations. There seem to be two factors that may be behind participants' increase satisfaction with the generated code. First, the increase in adoption of DSLs often suggests more use of customized generated code, which is more likely to match the developers' particular needs. Second, it is possible that modeling tools in general have improved their code generation.

We investigated the specific modeling tools that were reported in the survey as part of answers to a specific question (i.e. questions under topic 4: Platforms) as well as tools mentioned in the free text and open-ended questions. These tools include Papyrus, PlantUML, txtUML, MagicDraw, and others. It is

Topic 6: Problems with Model-Centric Approaches	Phase I			Phase II			Mean Gap
	% Slight Problem	% Bad Problem	Mean	% Slight Problem	% Bad Problem	Mean	
Models become out of date and inconsistent with code	**16.3**	**68.5**	**3.8**	**25**	**40.6**	**3.2**	**-0.6**
Models can not be easily exchanged between tools	26.4	51.6	3.3	19	40.7	3.3	0
Modeling tools are 'heavyweight'(install,learn,configure,use)	31.5	39.1	3.1	41	37.6	3	-0.1
Code generated from modeling tool not of the kind kind I would like	39.6	38.5	3	44	31.3	2.7	-0.3
Cannot model in enough detail-must write code	43.8	36	2.8	47	28.1	2.6	-0.2
Creating and editing model is slow	43.5	22.8	2.7	38	34.4	3	0.3
Modeling tools change, models become obsolete	44.6	32.6	2.7	31	34.4	3	0.3
Modeling tools lack features I need or want	44.9	21.3	2.6	44	18.8	2.6	0
Modeling tools hide too many details(fully visible in source)	44.6	23.9	2.6	34	31.3	2.9	0.3
Modeling tools are too expensive	46.7	26.7	2.6	38	15.7	2.7	0.1
Modeling tools cannot be analyzed as intended	51.1	25.6	2.5	56	21.9	2.5	0
Semantics of models different from prog. language	56.7	23.3	2.4	48	16.2	2.5	0.1
Modeling languages are not expressive enough	54.9	17.6	2.4	50	15.7	2.5	0.1
Modeling languages are hard to understand	62.6	9.9	2.2	58	15.2	2.3	0.1
Have had bad experience with modeling	63.7	16.5	2.2	61	16.2	2.2	0
Do not trust companies will continue to support their tools	**67.4**	**10.1**	**2**	**41**	**15.7**	**2.6**	**0.6**

Table 17 Problems with model centric approach

Topic 6: Problems with Code-Centric Approaches							
Potential problems	**Phase I**			**Phase II**			**Mean Gap**
	% Slight Problem	**% Bad Problem**	**Mean**	**% Slight Problem**	**% Bad Problem**	**Mean**	
Hard to see overall design	13.8	66	3.8	12	67.7	3.8	0
Hard to understand behaviour of system	19.1	60.6	3.6	30	45.4	3.2	-0.4
Code becomes of poorer quality over time	28.3	55.4	3.4	34	43.8	3.2	-0.2
Too difficult to restructure system when needed	22.6	51.6	3.4	22	53.1	3.4	0
Difficult to change code without adding bugs	**22.6**	**50.5**	**3.4**	**38**	**40.6**	**2.9**	**-0.5**
Changing code takes too much time	39.4	27.7	2.8	44	21.9	2.6	-0.2
Our prog. language leads to complex code	51.1	20.2	2.5	41	28.1	2.7	0.2
More skill than available to develop high quality code	53.8	22	2.5	47	31.3	2.7	0.2
Prog. Languages not expressive enough	64.8	14.3	2.1	69	18.8	1.9	-0.2
Organization culture does not like code-centric	72.8	14.1	1.9	78	6.3	1.7	-0.2
Our prog. language likely to become obsolete	**75.3**	**9.7**	**1.9**	**56**	**28.1**	**2.4**	**0.5**

Table 18 Problems with code centric approach

Software modeling you have done in the last year	%
Less than 10 hours	0.0
More than 10 hours but less than 40	25.0
More than 40 hours but less than 200	31.3
More than 200 hours	43.8

Table 19 Profiling information

possible that code generation improvements in these tools are reflected in participants' responses.

Another important trend is the increase in recognition that programming languages and related technologies and platforms could become quickly obsolete. This trend is particularly positive as it is a motivation for adopting model-centric approaches that tend to provide better support for platform independence. This claim is confirmed by the practitioners' perception in phase III post-survey study. Further, recent studies by Aldrich et al. supports this claim by adopting model-based development in robotics software projects where it can be more easily adapted to different hardware, tasks, and environments (Aldrich et al. 2019). They reported that models enable robots to automatically explore potential adaptations to the system architecture and code. This indicates that model-based approach and software modeling is adopted in cross-platform positively.

Downward Trends. There is a significant decrease in participants' satisfaction with the modeling tools they have or are using, as follows. More participants reported that 1) modeling tools are less capable of supporting communications with other developers and designers; 2) there is inadequate support for maintaining code and model in sync; 3) there is inadequate support for prototyping, and 4) modeling tools are not suitable for creating software designs in general. This is also reflected in a lower frequency of using and reusing models created by other developers.

There is a decrease in participants' satisfaction with modeling tools as evident by the data showing more participants find modeling tools to be overly complex, to require a significant learning curve, and to be difficult to use (phase III data). These issues are also reflected in the open-ended survey questions. The decrease in satisfaction with modeling tools is recurrent in the literature. A few recent studies have uncovered similar trends. Most notably, Anger et al. conducted a survey of the use of modeling tools and found that mostly used modeling tools lack feedback, being slow to use, difficulty drawing the diagrams, not interacting well with other tools, and being complex to use (L. T. Agner et al. 2019).

Eclipse, the open-source development platform, demonstrated a significant decline in use by the survey participants. UML version 1.* has decreased significantly, as to be expected due to more participants using the newer UML versions which are reflected in Table 10.

A large majority of the negative trends seem to be related to the perception of modeling tools in terms of usability and suitability for performing many tasks and activities. More participants find modeling tools increasingly difficult to learn and report the learning curve to be a significant challenge confirmed by phase III survey results. This analysis is also supported by Liebel et al. who reported on a modeling tool usability survey and found that students' perception tends to be more negative when the tool provides negative feedback, such as compiler errors in the generated code (Liebel et al. 2017).

There is also a general decline in the perception of modeling tools support for activities that involve communication and collaboration with others which reflects in Table 14. This may explain why more participants reported creating models upon request only and often using a pen and paper and whiteboards as a modeling platform. Moreover, responses indicate less re-use of existing models. This dissatisfaction with modeling tools is repeatedly mentioned in the free text responses across all three phases of survey data. Many participants argued that time investments in model creation and maintenance are not justified. Others argued that casual informal modeling (often without

Q	Question Text	SA +A	N	DA + SD	Mean
1	It is a significant concern to me that the modeling tools I use may not continue to be supported in the future.	81.3	12.5	6.3	4.3
2	It is a significant concern to me that the some of the programming languages I use may become obsolete.	81.4	11.8	12.5	3.9
3	It is important to be able to reuse modeling artifacts.	50.1	31.3	18.8	3.5
4	It is important to be able to work with models in brainstorming sessions.	81.3	12.5	6.3	4.2
5	Modeling tools are generally difficult to learn.	6.3	50.0	43.8	2.4
6	Modeling tools are generally difficult to use productively for software development.	25.0	43.8	31.3	2.9
7	Modeling tools are generally not good at communicating information about software to other team members.	25.0	25.0	50.1	2.6
8	Modeling tools in general do not provide adequate support for generating executable systems.	18.8	56.3	25.0	2.8
9	Code generated by modeling tools is generally not reusable.	25.1	25.0	50.1	2.8
10	Modeling tools are most effective in the requirements phase.	43.8	31.3	25.1	3.2
11	I have significantly increased my use of formal modeling using languages such as OCL.	43.8	31.3	25.1	3.2
12	I have significantly increased my use of domain-specific languages in the last few years.	50.1	31.3	18.8	3.4
13	I have significantly increased my use of code generation in the last few years.	50.1	25.0	25.1	3.4
14	My colleagues and I make significant use of informal drawing of models by hand, e.g. on whiteboards.	37.5	43.8	18.8	3.3
15	Synchronization between models and code has become less of a concern for users of modeling tools.	26.7	66.7	6.7	3.3

Table 20 Post survey study summary

using a modeling tool) is much more effective.

The data also shows a trend of a declining number of participants reporting performing the task of transcribing a model from an informal source (such as the whiteboard) to a modeling tool which is further supported by Silva et al. (Da Silva 2015b). The author implied that models are not just documentation artifacts, but also central artifacts in the software engineering field, allowing the creation or automatic execution of software systems starting from those models. This could be interpreted as both positive and negative perceptions. The positive interpretation is that more participants are creating the models on modeling tools, and hence not requiring the transcribing task. Alternatively, this trend can be interpreted negatively in that participants do not find the modeling tool flexible enough to support their need for sketching and creating models in an agile and quick fashion.

6.2. Expected and Unexpected Trends

The primary expected trend is the decline in the use of older UML versions. This was expected since the two phases of this study span a significant period of time. This result suggests that respondents were careful in their replies and increases our confidence in the validity of the results. Other expected results in the continuing dissatisfaction with modeling tools capabilities, usability, and support for collaboration and communication also confirmed by the phase III post-study survey qualitative data. This result was expected to us based on our personal experiences with a broad range of software modeling and design tools.

The first unexpected result is an increase in the practices of modeling and design despite the decrease in participant's satisfaction with modeling tools. The increase in the practices of modeling is muted, especially when compared to the increase in the size and complexity of the software being developed. While this is an unexpected result, it is one that this study highlights; practitioners' adoption of modeling and design practices is not on par with the increase in code size and its complexity. Studies found that larger projects might be expected to make wider use of UML diagram types, but this is generally not the case (Anda et al. 2006b). Further, Anda et al. reported that when the system size and complexity increase the associated cost of UML modeling is also elevated (Anda et al. 2006b).

Another important unexpected result is the sizable decline in satisfaction with modeling tools especially with respect to their support for communication and collaboration. This highlights the need for a broad and diverse set of design tools, from lightweight informal modeling to fully-fledged model-centered tools.

Further, unexpected result is related to the increase in the adoption of formal modeling and domain-specific modeling languages (DSLs) even though we sought participants from development venues where we wouldn't expect that DSLs and formal modeling to be broadly adopted. This result is consistent with all three survey phase data. We interpret this that practitioners tend to value software design and modeling more favorably depending on how the models are deployed productively in and throughout their development endeavors. Clearly, formal models contribute to test and code generation, and DSLs tend to be uniquely designed to the developers' domains. We also interpret this unexpected result by the increase in the availability and capability of DSL development platforms. These platforms reduce barriers for the development of usable DSLs and broaden the user-base of such specialized development technologies.

6.3. Persistent Challenges and Emerging Opportunities

We identify persistent challenges as those issues in modeling and design that are found to be problematic in all the phases with little or no improvement. Many of these challenges manifest themselves in modeling tools' complexity, their required learning curve, and their inadequate support for flexibility.

Another persistent challenge is related to the practice of developing executable models or relying on models to generate all code and executable artifacts. This challenge, despite being slightly reduced, remains a significant limiting factor for broader adoption. The need to modify the generated code because it is incomplete, or because of the need to add functionality introduces a whole set of challenges, including managing the synchronization between code and models. Model-generated code has side effects that are discussed in the study by He et al. who investigated code quality and technical debt of model based projects (He et al. 2016). The authors in this study (He et al. 2016) demonstrated that MDE code generators incur more associated cost than handwritten code. This influences significantly the maintenance of model-based software development practices.

A third challenge is the limited scope of modeling activities in terms of their occurrences in the project lifecycle, or along with the various activity disciplines. It is only in the requirements phase that more than half of the participants reported using modeling frequently. Across other activities, modeling remains rather low. Moreover, almost all of the potential problems with model-centric approaches identified by participants have remained or declined only slightly.

A fourth challenge that emerged and also confirmed from the phase III survey qualitative data is the need to perform tool customization. We infer that the needed customization is beyond what tools offer. This is often a challenge for tool developers; more customization available for end-user often results in an increase in tool complexity that often impedes broader adoption.

A fifth and important challenge pertains the lack of support from managers and senior developers for adopting modeling tools. Participants in this study argue that modeling tools incur significant upfront costs and the pay-off materialize often at maintenance and post-deployment phases. As such, it is increasingly more difficult to build a business case to convince leaders and senior developers to invest in tool and technology acquisition.

The key emerging opportunity is the significant increase in modeling activities in general, and particularly the increase in the adoption of Domain-Specific Modeling Languages and formal modeling.

7. Threats to Validity

The primary threats to validity of this study are summarized below. We have also outlined the steps we have taken to help mitigate these threats.

Question interpretation. Respondents may have misunderstood the intended meaning of our questions. We took two steps to reduce the ambiguity of the questions. First, five independent researchers reviewed the survey (phase II) structure, wording, and questions. Second, we piloted the survey and reviewed any ambiguities, and implemented the suggested comments. Both activities helped to improve the overall survey prior to go-live. Since the primary goal of this study is to uncover trends, we expect that bias inherits in the survey to be present in both phases and its effects would be largely minimized.

Researcher bias. Many of the survey questions attempt to uncover trends related to both model-centric and code-centric approaches. A potential bias could be introduced if our survey appeared to be overly negative towards either modeling or software coding. To reduce the chance of bias we aimed to be objective whenever we referring to both code-centric and model-centric questions, as well as presenting the questions in random order. We also maintained the same questions and wording for the two phases to minimize the effects of any potential bias. We also piloted the survey and collected feedback from the participants. We revised the wording to ensure consistent interpretation of questions.

Non-randomized sample and representation. To help ensure that our sample was based on a representative collection of software practitioners, we approached both open and closed forums for participation. In particular, we submitted survey links to Digg.com, and Dzone.com - two popular technology and news sites. We submitted email requests to UML user groups, agile user groups, Java user groups, and process user groups. Our demographics results indicate that we do have representation from most regions of the world, most educational backgrounds, most software industries, and most types of developers. However, the sample size of Europe-based participants is relatively small. Therefore, we do not claim that the list of participants' domain and demographics is complete. We also conducted extensive sub-sample analysis that included analyzing sub-samples of participants, their educational and experience profiles, as well as sub-sample analysis of software application domains to ensure adequate representation.

A related threat pertains to self-selection. Those who opt to participate in the study may be already those who are enthusiastic about software design and modeling. We consistently recruited participants in the two phases. Therefore, any bias in selection would have an equal or similar impact. This impact would be canceled out since the results and analysis are based on gaps between the two phases of the study.

The sample size of phase III post-study survey is relatively small. The goal of the post-study survey is not to uncover additional trends, but rather to clarify and help interpret the trends uncovered by the study. Secondly, the post-study survey included significant open-ended questions and would require significant time from participants to complete the study.

Participant Fatigue. This survey study included about 18 questions many with multiple choices and open-ended questions with free text. We estimate that the survey takes about one hour to fully answer all questions. Hence, there is a risk of participant fatigue which may affect data validity. We did the following to minimize this risk. First, the survey and our solicitations clearly stated the expected duration of the survey. Second, we included many cues in the survey to inform the participant of progress and the remaining sections. Third, we allowed participants to skip sub-questions based on their responses. Fourth, we only included complete responses in the data analysis.

8. Conclusion

This paper reports on a survey conducted on three phases ten years apart. The goal of the survey is to contribute to uncovering trends in the practices of software design and modeling. The survey solicited 248 participants and included 18 questions.

The survey analysis characterizes trends in the practice, including upward, downward, positive, negative, and unexpected trends. The survey suggests some level of increase in the adoption of the broad practices of modeling as well as an increase in the adoption of domain-specific and formal modeling. We also observed an increase in the practices of casual modeling using whiteboards and pen and paper approaches. The data also suggests a persistent dissatisfaction with software modeling tools. Participants find modeling tools to be inadequate in their support for collaboration and communication. Participants also consistently reported inadequacy of the generated code for their development purposes and needs. Many participants argued that modeling cannot be justified due to the learning curve, modeling tool complexity, and inadequacy for delivering executable artifacts. The phase III post-survey study confirms the main findings of the study and highlights concerns about future support for both modeling tools and development platforms.

The three phases of the survey draw a picture of the practices and how they are changing over time. Our goal is to help researchers, educators, practitioners, and modeling tools providers to focus on important aspects of relevance to the profession.

References

Agarwal, R., De, P., Sinha, A. P., & Tanniru, M. (2000). On the usability of oo representations. *Communications of the ACM*, *43*(10), 83–89.

Agarwal, R., & Sinha, A. P. (2003). Object-oriented modeling with uml: a study of developers' perceptions. *Communications of the ACM*, *46*(9), 248–256.

Agner, L. T., Lethbridge, T. C., & Soares, I. W. (2019). Student experience with software modeling tools. *Software & Systems Modeling*, *18*(5), 3025–3047.

Agner, L. T. W., Soares, I. W., Stadzisz, P. C., & SimãO, J. M. (2013). A brazilian survey on uml and model-driven practices for embedded software development. *Journal of Systems and Software*, *86*(4), 997–1005.

Aldrich, J., Garlan, D., Kästner, C., Le Goues, C., Mohseni-Kabir, A., Ruchkin, I., … others (2019). Model-based adaptation for robotics software. *IEEE Software*, *36*(2), 83–90.

Anda, B., Hansen, K., Gullesen, I., & Thorsen, H. K. (2006a). Experiences from introducing uml-based development in a large safety-critical project. *Empirical Software Engineering*, *11*(4), 555–581.

Anda, B., Hansen, K., Gullesen, I., & Thorsen, H. K. (2006b). Experiences from introducing uml-based development in a large safety-critical project. *Empirical Software Engineering*, *11*(4), 555–581.

Arisholm, E., Briand, L. C., Hove, S. E., & Labiche, Y. (2006). The impact of uml documentation on software maintenance: An experimental evaluation. *IEEE Transactions on Software Engineering*, *32*(6), 365–381.

Badreddin, O., Khandoker, R., Forward, A., Masmali, O., & Lethbridge, T. C. (2018). A decade of software design and modeling: A survey to uncover trends of the practice. In *Proceedings of the 21th acm/ieee international conference on model driven engineering languages and systems* (pp. 245–255).

Badreddin, O., Lethbridge, T. C., & Elassar, M. (2013). Modeling practices in open source software. In *Ifip international conference on open source systems* (pp. 127–139).

Badreldin, O., Lethbridge, T., Sturm, A., Dixon, W., Hamou-Lhadj, A., & Simmons, R. (2015). The effects of education on students' perception of modeling in software engineering..

Budgen, D., Burn, A. J., Brereton, O. P., Kitchenham, B. A., & Pretorius, R. (2011). Empirical evidence about the uml: a systematic literature review. *Software: Practice and Experience*, *41*(4), 363–392.

Cherubini, M., Venolia, G., DeLine, R., & Ko, A. J. (2007). Let's go to the whiteboard: how and why software developers use drawings. In *Proceedings of the sigchi conference on human factors in computing systems* (pp. 557–566).

Da Silva, A. R. (2015a). Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, *43*, 139–155.

Da Silva, A. R. (2015b). Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, *43*, 139–155.

Dobing, B., & Parsons, J. (2006). How uml is used. *Communications of the ACM*, *49*(5), 109–113.

Dobing, B., & Parsons, J. (2008). Dimensions of uml diagram use: a survey of practitioners. *Journal of Database Management (JDM)*, *19*(1), 1–18.

Dreamincode. (2017a). Retrieved from http://www.dreamincode.net/forums/

Elaasar, M., Noyrit, F., Badreddin, O., & Gerard, S. (2017). Reducing uml modeling tool complexity with architectural contexts and viewpoints..

Forward, A., & Lethbridge, T. C. (2008). Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals. In *Proceedings of the 2008 international workshop on models in software engineering* (pp. 27–32).

Garousi, V., Coşkunçay, A., Betin-Can, A., & Demirörs, O. (2015). A survey of software engineering practices in turkey. *Journal of Systems and Software*, *108*, 148–177.

He, X., Avgeriou, P., Liang, P., & Li, Z. (2016). Technical debt in mde: a case study on gmf/emf-based projects. In *Proceedings of the acm/ieee 19th international conference on model driven engineering languages and systems* (pp. 162–172).

Hutchinson, J., Whittle, J., Rouncefield, M., & Kristoffersen, S. (2011). Empirical assessment of mde in industry. In *Proceedings of the 33rd international conference on software engineering* (pp. 471–480).

Iivari, J. (1996). Why are case tools not used? *Communications of the ACM*, *39*(10), 94–103.

Javaforum. (2017b). Retrieved from https://www.java-forums.org/forum.php

Javaranch. (2017c). Retrieved from https://javaranch.com/

Khelladi, D. E., Combemale, B., Acher, M., & Barais, O. (2020). On the power of abstraction: a model-driven co-evolution approach of software code. In *Proceedings of the acm/ieee 42nd international conference on software engineering: New ideas and emerging results* (pp. 85–88).

Kleppe, A. G., Warmer, J., Warmer, J. B., & Bast, W. (2003). *Mda explained: the model driven architecture: practice and promise*. Addison-Wesley Professional.

LaToza, T. D., Venolia, G., & DeLine, R. (2006). Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on software engineering* (pp. 492–501).

Liebel, G., Badreddin, O., & Heldal, R. (2017). Model driven software engineering in education: A multi-case study on perception of tools and uml. In *Software engineering education and training (csee&t), 2017 ieee 30th conference on* (pp. 124–133).

Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M. A., Nordmoen, B., & Fritzsche, M. (2013). Where does model-driven engineering help? experiences from three industrial cases. *Software & Systems Modeling*, *12*(3), 619–639.

Professional coding and modeling practices,. (2017). Retrieved from https://goo.gl/bQV9Ph

Professional coding and modeling practices: Post survey study,. (2018). Retrieved from https://goo.gl/TbYmUY

Technical report tr-2018-01: A decade of software design practices: A survey to uncover trends of the practice. accessed july 2018. (2017). Retrieved from https://figshare.com/s/b54ca33b8717c0fa1d3d

Technical report tr-2019-01: The evolution of software design practices over a decade: A long term study of practitioners. accessed july 2018. (2020). Retrieved from https://doi.org/10.6084/m9.figshare.12593270

Whittle, J., Hutchinson, J., & Rouncefield, M. (2014). The state of practice in model-driven engineering. *IEEE software*, *31*(3), 79–85.

About the authors

Omar Badreddin is a software design professional and researcher. He is a faculty member in Computer Science at University of Texas. He has authored several books and scientific articles on software design and reengineering. He has contributed to many prominent open and closed source software that has reached global user base, including the Eclipse platform. He is an active open source contributor and an advocate for sustainable software engineering practices. Dr. Badreddin is the primary author of Susereum (www.Susereum.com), the blockchain platform that establishes immutable credit for code authors and contributors. You can contact him at obbadreddin@utep.edu or visit https://badreddin.com.

Khandoker Rahad is a Ph.D. candidate in computer science at the University of Texas. He is working as a research and PhD teaching assistant in the computer science department (2016-2020).His research interest lies in the area of software design, UML modeling and open source mining. His current research focuses on investigating the practices of software designs to uncover patterns of practices, and systematically evaluate the potential benefits of model-driven development. This includes investigating open source repositories, analyzing code bases, and conducting surveys to better understand current practices and uncover any emerging trends. You can contact him at rahadiit@gmail.com.

Andrew Forward is a part-time professor at the University of Ottawa teaching a wide range of topics on software engineering, computer science and project management. Dr. Forward's research interests are in software quality, automation, documentation and software modelling. Andrew also works as a Software Engineer at CrossFit, a world leading platform for health, happiness and performance. You can contact him at aforward@uottawa.ca.

Timothy Lethbridge is a vice dean and professor of software engineering and computer science at the University of Ottawa, where he has taught for 30 years. His research focuses on usability of software engineering tools, including his own tool Umple, as well as on software engineering education and enterprise architecture. He has published 150 scientific papers and one textbook on software engineering. He is a professional engineer and has held several volunteer roles in the IEEE and the Canadian Information Processing Society. He received the IEEE Computer Society TCSE Outstanding Educator Award in 2016. You can contact him at timothy.lethbridge@uottawa.ca or visit https://uniweb.uottawa.ca/members/119/profile?embed=2.