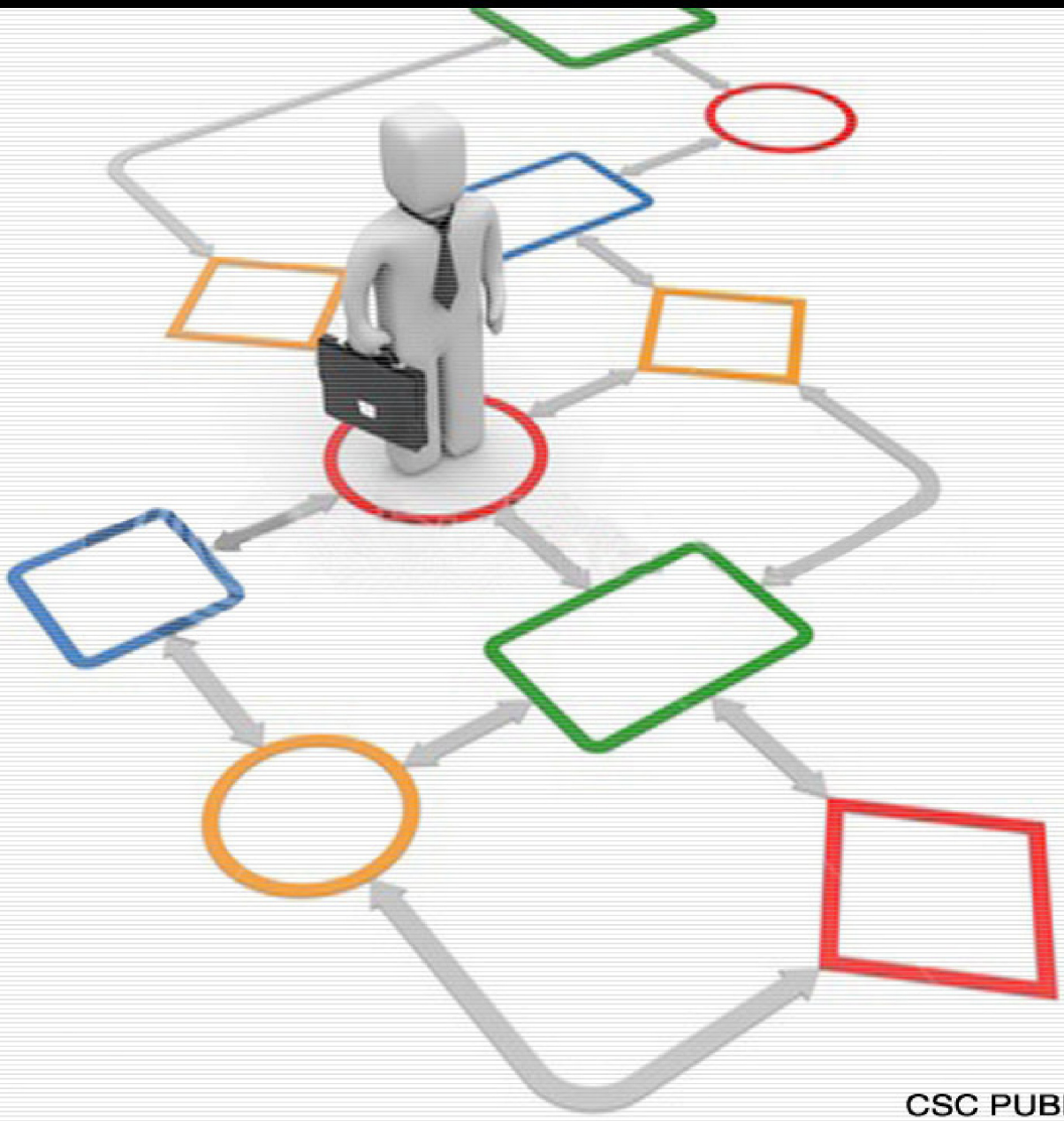
Volume 9 ▪ Issue 2 ▪ October 2021

INTERNATIONAL JOURNAL OF

SOFTWARE ENGINEERING (IJSE)

ISSN : 2180-1320

Publication Frequency: 6 Issues / Year



CSC PUBLISHERS
http://www.cscjournals.org

Copyrights 2021. All rights reserved. Computer Science Journals (CSC Journals).

INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE)

VOLUME 9, ISSUE 2, 2021

EDITED BY
DR. NABEEL TAHIR

ISSN (Online): 2180-1320

I International Journal of Software Engineering (IJSE) is published both in traditional paper form and in Internet. This journal is published at the website https://www.cscjournals.org, maintained by Computer Science Journals (CSC Journals), Malaysia.

IJSE Journal is a part of CSC Publishers
Computer Science Journals
https://www.cscjournals.org

INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE)

Book: Volume 9, Issue 2, October 2021
Publishing Date: 31-10-2021
ISSN (Online): 2180-1320

This work is subjected to copyright. All rights are reserved whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illusions, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication of parts thereof is permitted only under the provision of the copyright law 1965, in its current version, and permission of use must always be obtained from CSC Publishers.

IJSE Journal is a part of CSC Publishers
https://www.cscjournals.org

© IJSE Journal
Published in Malaysia

Typesetting: Camera-ready by author, data conversation by CSC Publishing Services – CSC Journals, Malaysia

CSC Publishers, 2021

EDITORIAL BOARD

EDITORIAL BOARD MEMBERS (EBMs)

Dr Xiaohong (Sophie) Wang
Salisbury University
United States of America

Dr. Amit Mishra
Baze University
Nigeria

Dr. AJAYI Olusola Olajide
Adekunle Ajasin University
Nigeria

TABLE OF CONTENTS

Volume 9, Issue 2, October 2021

Pages

27 – 62 The Application of Function Models In Software Design: A Survey Within the Software Community
Pujani Abayatilake, Lucienne T.M. Blessing

EDITORIAL PREFACE

The International Journal of Software Engineering (IJSE) provides a forum for software engineering research that publishes empirical results relevant to both researchers and practitioners. It is the *Second* Issue of Volume *Nine* of IJSE and it is published bi-monthly, with papers being peer reviewed to high international standards.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Started with Volume 9, 2021, IJSE appears with more focused issues. Besides normal publications, IJSE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

IJSE encourage researchers, practitioners, and developers to submit research papers reporting original research results, technology trend surveys reviewing an area of research in software engineering, software science, theoretical software engineering, computational intelligence, and knowledge engineering, survey articles surveying a broad area in software engineering and knowledge engineering, tool reviews and book reviews. Some important topics covered by IJSE usually involve the study on collection and analysis of data and experience that can be used to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies. IJSE is a refereed journal that promotes the publication of industry-relevant research, to address the significant gap between research and practice.

IJSE gives the opportunity to researchers and practitioners for presenting their research, technological advances, practical problems and concerns to the software engineering. IJSE is not limited to a specific aspect of software engineering it cover all Software engineering topics. In order to position IJSE amongst the most high quality journal on computer engineering sciences, a group of highly professional scholars are serving on the editorial board. IJSE include empirical studies, requirement engineering, software architecture, software testing, formal methods, and verification.

International Editorial Board ensures that significant developments in software engineering from around the world are reflected in IJSE. The submission and publication process of manuscript done by efficient way. Readers of the IJSE will benefit from the papers presented in this issue in order to aware the recent advances in the Software engineering. International Electronic editorial and reviewer system allows for the fast publication of accepted manuscripts into issue publication of IJSE. Because we know how important it is for authors to have their work published with a minimum delay after submission of their manuscript. For that reason we continue to strive for fast decision times and minimum delays in the publication processes. Papers are indexed & abstracted with International indexers & abstractors.

Editorial Board Members
International Journal of Software Engineering (IJSE)

The Application of Function Models In Software Design: A Survey Within the Software Community

Pujani Abayatilake *pujani_abayatilake@alumni.sutd.edu.sg*
Engineering Product Development
Singapore University of Technology and Design
487372, Singapore

Lucienne T.M. Blessing *lucienne_blessing@sutd.edu.sg*
Engineering Product Development
Singapore University of Technology and Design
487372, Singapore

Abstract

Numerous function modelling approaches exist for software design. However, there is little empirical evidence on how these approaches are used in the early stages of software design. This article presents the results of an online survey on the application of function models in the academic and industrial software development community. The results show that more than 90% of the 75 respondents agreed with the statement that software projects that use function modelling techniques have a higher chance of success than other projects. UML is the most widely accepted and used modelling approach among the respondents, but only a handful of UML diagrams appear to be prominently addressed during the early software design stages. Asked for reasons for selecting or rejecting UML models the majority of respondents mentioned using function models to understand software requirements and communicate these with clients and technical teams, whereas lack of familiarity, the time-consuming nature of some models and data redundancy are widely mentioned reasons for not or seldomly using certain models. The study also shows a strong relationship between model usage and respondents' professions. We conclude that improvements are required to ensure the benefits of the various available models and the links between the models can be fully exploited to support individual designers, to improve communication and collaboration, and to increase project success. A short discussion on the chosen solution direction - a simplified function modelling approach – closes the paper.

Keywords: Function Modelling, Unified Modelling Language (UML), Software Design, SDLC, Conceptual Design, Object-Oriented Modelling, Empirical Studies.

1. INTRODUCTION

"Function modeling is the name given to the activity of developing models of devices, products, objects, and processes based on their functionalities and the functionalities of their subcomponents." (Erden et al., 2008). Function models assist users in the design process from requirements to implementation, and according to Jacobson et al. (Jacobson, Booch, & Rumbaugh, 1999) they help to visualize the system components using diagrams over words, hence function modelling plays a vital role in the design process regardless of the discipline.

In software development function models are blueprints of the system and they help to better understand the system, especially complex software where the system cannot be imagined entirely (Booch, Rumbaugh, & Jacobson, 1998). Moreover, function modelling mitigates the issue of missing components in complex software while supporting the identification of possible errors before coding starts (Rumbaugh, Jacobson, & Booch, 1999). Especially in the software conceptual design stage, function models act as a common language between all the project respondents (Erden et al., 2008) and bridge requirements and system components. Today,

software function models such as UML and architecture models allow designers and developers to transform their software designs to programming languages (Atan, Ghani, Selamat, & Mahmod, 2007).

The introduction of function models in software development began around six decades ago, with the development of various software design paradigms and methods (Booch, 1994). These design paradigms were introduced to cope with the increasing complexity of software systems and the evolution of SDLC (Software Development Life Cycle) models (Capretz, 2003). SDLC describes the steps to follow when developing software (Kaur, 2017; Modi, Singh, & Chauhan, 2017). The key aim is to deliver quality software to the end customer within a given timeline and budget (Khan, Shadab, & Khan, 2020). Over the six decades, SDLC models have been evolving significantly; starting from structured stage-wise, iterative, and incremental to the agile models that are popular today (Al-Zewairi, Biltawi, Etaiwi, & Shaout, 2017). The key intention of software design paradigms was to analyze and organize the system requirements and functionality during the early design phases. The structured and object-oriented (OO) analysis and design paradigms are the two key software design paradigms in literature (Capretz, 2003; Rickman, 2001). Methods such as Data-Driven design (DD) and Specification and Descriptive Language (SDL), were design approaches used in the transition period from structured to OO design (Wieringa, 1998). For each design paradigm, several function modelling methods and models have been proposed (Wieringa, 1998).

The key difference between structured and OO function models is that structured models use *functional decomposition* while OO models use *object decomposition* to visualize the system (Wieringa, 1998). In other words, structured models distinguish behavioral and structural aspects of the system, i.e. functions and data respectively, using a hierarchical pattern to describe the functions, whereas OO models first identify the system as a set of objects and then define the system behavior using object relationships (Firesmith, 1991). Structured function models such as Data Flow Diagrams (DFD), activity diagrams (IDEF), and state transitions diagrams are popular for describing the behavior of the system, while Entity Relationship Diagrams (ERD) are widely used to show the data structure of the system (Baldwin, Austin, Hassan, & Thorpe, 1999), (Gregersen & Jensen, 1999), (Colquhoun, Baines, & Crossley, 1993), (Dorador & Young, 2000), and (Wieringa, 1998). OO function models are mainly described using UML (Unified Modelling Language) which allows incorporating all the OO concepts used throughout the design life cycle into a common modelling platform (Rumbaugh et al., 1999).

UML was started in 1994 by Booch and Rumbaugh, who integrated their Object-Oriented Design (OOD) and Object Modelling Technology (OMT) methods and added diagrams such as class, objects, collaboration, and state chart diagrams. In the mid-1990s, they included Jacobson's use case-driven method (*Objectory*) and the updated versions of their methods (Booch et al., 1998). Eventually, UML became the most widely adapted and standard function modelling approach because of its simplicity and suitability for complex, large, and interactive software systems (Jacobson et al., 1999) (Champeaux et al., 1990). To ensure the proper use of UML, a Rational Unified Process (RUP) framework was developed in the late 1990s (Anwar, 2014). RUP helps programmers to select the most suitable function models for each project. The latest version, UML 2.5, consists of 14 different function models.

At the time of the introduction of OO design concepts and UML in the 1990s, many organizations switched their projects from structured function models to UML based function models (Champeaux et al., 1990) due to the lack of compatibility of structured models with new programming languages, which were OO-based, and the lack of beneficial OO concepts like data abstraction, component reusability, and inheritance (Booch et al., 1998; Jilani, Usman, & Nadeem, 2011). However, some organizations are reluctant to switch from structured function models, despite UML-based OO models are more applicable from the technical perspective (Champeaux et al., 1990).

Even though UML is accepted as highly beneficial compared to the other traditional models, users have found various shortcomings in its application. Rickman, for example, mentions incompleteness and complexity as major obstacles (Rickman, 2001). He mentions that, as the models are built around objects, there is no single model that represents a complete functionality of the system, which can lead to missed requirements and - with information scattered over multiple function models - to inconsistency and ambiguity. As a result, the management of the models becomes complex and the totality less understandable (Grossman, Aronson, & McCarthy, 2005). According to Lange et al. (Lange, Chaudron, & Muskens, 2006), incomplete models prompt bad quality products or wrong products even after spending a high amount of testing effort. Chren et al. (Chren, Buhnova, Macak, Daubner, & Rossi, 2019) found that most students find it difficult to absorb UML due to its complexity, which is evident in the many mistakes in their modelling coursework or software projects.

Some researchers have suggested that UML models should be adapted to the current trend of programming to ensure that they suit the more flexible and faster pace of agile development (Ambler, 2004), (Kobryn, 2002). However, this has not been the case yet. Agile approaches have developed further. Several agile-specific models have been introduced recently such as user stories and storyboarding (Hvatum & Wirfs-Brock, 2018), (Cooper, Reimann, & Cronin, 2007), which can be considered types of function models. There are no equivalent UML models. Rumpe (Rumpe, 2017) attempts to bridge UML and agile approaches and argues that even though UML is designed for plan-based projects, it is adaptable to agile context because of its ability to represent complex content.

Based on the literature, we believe that the use of function models contributes to success. It seems, however, that programmers are still reluctant to use function models in early design stages and only use a limited number of function models, even in the most popular function modelling approaches like UML. Akdur et al. (Akdur, Garousi, & Demirörs, 2018) found that the majority of practitioners use sketches or informal modelling languages, while very few use formal function modelling languages. The reason seemed to be that many practitioners believe that function models are not important for communication or for understanding the software system and only useful for auto-generating software artifacts. According to Dobing and Parsons (Dobing & Parsons, 2008), the primary reason – at the time - for not using OO function models was the lack of familiarity with OO concepts. Whether the reason for not using structured functional models is similar is not clear, as Dobing and Parsons only studied OO models. Farias et el. (Farias, Gonçales, & Bischoff, 2018) studied the use of UML models and found that “there is still a lack of understanding regarding practices and perceptions of UML use from the perspective of practitioners.” Given the importance of function models, this places the whole design process at risk and is likely to affect the success of the product. They also found that 62% of their participants reported that “the automatic creation of UML diagrams to represent a big picture of the system under development, would be useful to boost UML use”.

The survey described in this publication was primarily conducted to obtain a good understanding of the *current* application of both structured and OO function models and the reasons for using and not using certain approaches and models.

The survey was part of a research project aimed at identifying the features and issues of the function models widely used in industry (the focus of this publication) and develop a new function modelling approach that integrates the beneficial features of different models into one framework while mitigating the identified issues. This new approach, which includes a variety of function models from different design paradigms, is expected to encourage the use of function models in the early design phases and, in doing so, contribute to the development of quality software and project success.

The remainder of the paper is organized as follows. Section 2 presents empirical literature on the various function modelling approaches. Section 3 introduces the research questions, the research methodology and the survey design. The findings of the survey are presented in Section 4 while

the summary of the findings and their consequences in developing an effective and easy-to-use integrated function modelling approach are discussed in Section 5. Section 6 concludes the paper.

2. RELATED WORK

A considerable amount of research has investigated the practical applications of UML (Dobing & Parsons, 2008; Erickson & Siau, 2007; Grossman et al., 2005; Petre, 2013; Reggio, Leotta, & Ricca, 2014; Reggio, Leotta, Ricca, & Clerissi, 2013; Wrycza & Marcinkowski, 2007). However, only few have addressed the traditional modelling approaches, such as structured and data-driven (DD) models (Agarwal, De, & Sinha, 1999; Falessi, Cantone, & Grande, 2007; Wieringa, 1998). Even though UML is generally considered the most widely used modelling language, almost all the UML-related studies specifically mention the limitations discussed in Section 1.

A survey in 2004 (Grossman et al., 2005) on the use of UML among 131 IT practitioners from organizations in 32 countries, assessed UML criteria such as accuracy, level of detail, understandability, complexity, flexibility, etc. The authors found that UML is not a complete solution to design complex software systems and lacks the constructs for systems engineering and good architecture work. However, they did find that UML is the best way to communicate with the teams to determine enough about the solutions to begin writing tests and code. Almost 10 years later Petre (Petre, 2013) conducted interviews into the use of UML in industry, involving 50 professional software engineers in 50 companies across the globe. He found that 20 years after the introduction of UML, 70% of the respondents still do not use UML at all and 22% use UML infrequently, selectively, and in an informal way. As far as UML was used, only few diagrams were used: class, sequence, activity, state chart, and use case diagrams. Respondents mentioned using UML for: requirement elicitation, concept analysis of a domain, modelling concurrency, and as a “Thought tool” to start coding. The barriers they mentioned included: no benefits over current practices such as flow charts and ERD, lack of context, lack of consistency, and difficulty in understanding notations.

The series of questionnaires by Erickson and Siau (Erickson & Siau, 2007) in 2003-2004 involved 29 respondents (22 from academia – 9 of which were students - and 7 practitioners) and was aimed at identifying the subset (kernel) of UML for three different UML application areas (real-time, web-based, and enterprise) to inform the training and usage of UML. They found that class, use case, and sequence diagrams were the top 3 ranked models for each of the application areas, even though the rank order in each application area differed. In 2007 Wrycza and Marcinkowski (Wrycza & Marcinkowski, 2007) too carried out a survey to identify the minimum set of UML 2.X diagrams required for UML teaching. 180 university students who took UML courses were asked to evaluate UML against several criteria, such as complexity level, user-friendliness, usefulness, and effectiveness in code generation. 91% of the students considered UML to be complex, because of the many diagrams and notations. The students found class, use case, activity, and sequence diagrams the most useful for modelling, and almost all the behavioral and interactive diagrams more user-friendly compared to the structural diagrams, except for the class diagram. According to the authors, “Due to the pragmatic role of class diagrams for programming, they have also achieved a high rank of acceptance.” The students also found UML not very useful for code generation. Based on the findings the authors concluded that UML use case, class, activity, and sequence diagrams are the minimal set of UML2.X.

A survey by Reggio et al. (Reggio et al., 2014) in 2013-2014 among 275 respondents (60% practitioners, 40% from academia) aimed at understanding UML knowledge and usage in industry and academia. They found use case, class, state, sequence, and activity diagrams as widely known and used in practice. Those in academia were found to have more knowledge about UML than practitioners, and that overall, the level of knowledge was positively related to working experience. Among the practitioners, developers were the ones who least used UML, while business analysts and software designers were the ones who used UML most. They, therefore, concluded that UML is less applicable for the later development phases. Dobing and Parsons (Dobing & Parsons, 2008) carried out a comprehensive survey in 2003-2004 among 284 analysts

from different UK industries in which they compared 7 UML diagrams including use case narratives. Their objective was to study how often each diagram was being used and the reasons to use or avoid these. The study showed that analysts use almost every diagram to validate requirements and communicate within project teams. Class, sequence, and use case diagrams were the most used ones, the collaboration diagram was the least used. The results further showed that even though UML is often presented with a use case-driven approach, only 44% of the respondents utilized use case narratives in their projects. Reasons given by the respondents for not using the 7 UML diagrams are: class diagram – not well understood; sequence, use case, and activity diagrams – insufficient value to justify the cost; collaboration diagram – redundant when using sequence diagrams; use case narratives – not useful for communicating with programmers; and state diagram – not useful for most projects.

Although all the above-mentioned surveys provide valuable insights, they focus on UML only. Only a few, older studies were found that compare structured and OO modelling (Agarwal et al., 1999; Falessi et al., 2007). Agarwal et al. (Agarwal et al., 1999) conducted laboratory experiments in 1999 to compare comprehension of structured and OO models among 71 undergraduate students majoring in information systems. In most cases they observed no significant difference between these models. However, for more complex problems, structured models seemed to be easier to understand and more efficient than OO models. Their explanation was that students seem to prefer process flows above object-driven models. It has to be noted that this study took place over 20 years ago and these findings may no longer be valid. Falessi et al. (Falessi et al., 2007) conducted laboratory experiments, involving 50 master students, to compare the time required to produce SADT (Structured Analysis and Design Technique) models and UML diagrams with RUP (Rational Unified Process) in the early software design phase. They introduced two scenarios: a new project and the enhancement of an existing project. No significant differences were observed , but according to the authors, it appeared that OO models are more user sensitive and have advantages like reusability over structured models.

The studies described in the literature identify advantages and limitations of the different function models, but they mainly assess UML diagrams rather than other function models, and several of these took place some years ago. We aim to fill this gap by providing empirical evidence on the current usage of the various function modelling approaches used in practice and identify their experienced and perceived benefits and shortcomings.

3. RESEARCH METHODOLOGY

The study reported here consisted of an online survey conducted over a period of six weeks from February to March of 2020 among practitioners and people in academia. The following research questions were addressed.

RQ1. Which function models are used for software design? (personally, as well as within the organization)
RQ2. How are these function models applied during conceptual design?
RQ3. How important are the function models in the conceptual design phase?
RQ4. What are the reasons for using function models in software projects?
RQ5. What are the reasons for not using function models in software projects?
RQ6. What are the currently used SDLC models (personal and organizational)? Do the personal SDLC models have an influence on function model usage?
RQ7. In which phases of software development are function models considered to be most important?
RQ8. How does the use of function models affect the success of a project?

3.1 Questionnaire Design

The questionnaire contains both multiple-choice and open-ended questions to best capture the respondents' opinions. The 25 questions are based on the above research questions and structured as shown in FIGURE 1. Questions Q1-Q6 ask for general information such as the profession, experience, the number of software projects in which the respondents were involved,

and their use of software development methodologies. The main body of the questionnaire (Q7-Q23) addresses the research questions. Which questions are asked depends on a respondent's use of function models and their experience with UML and other function modelling approaches for conceptual design. The last two questions Q24-Q25 are the same for all respondents, asking for their general opinion on the importance of function models in software development and their impact on the success of software products.

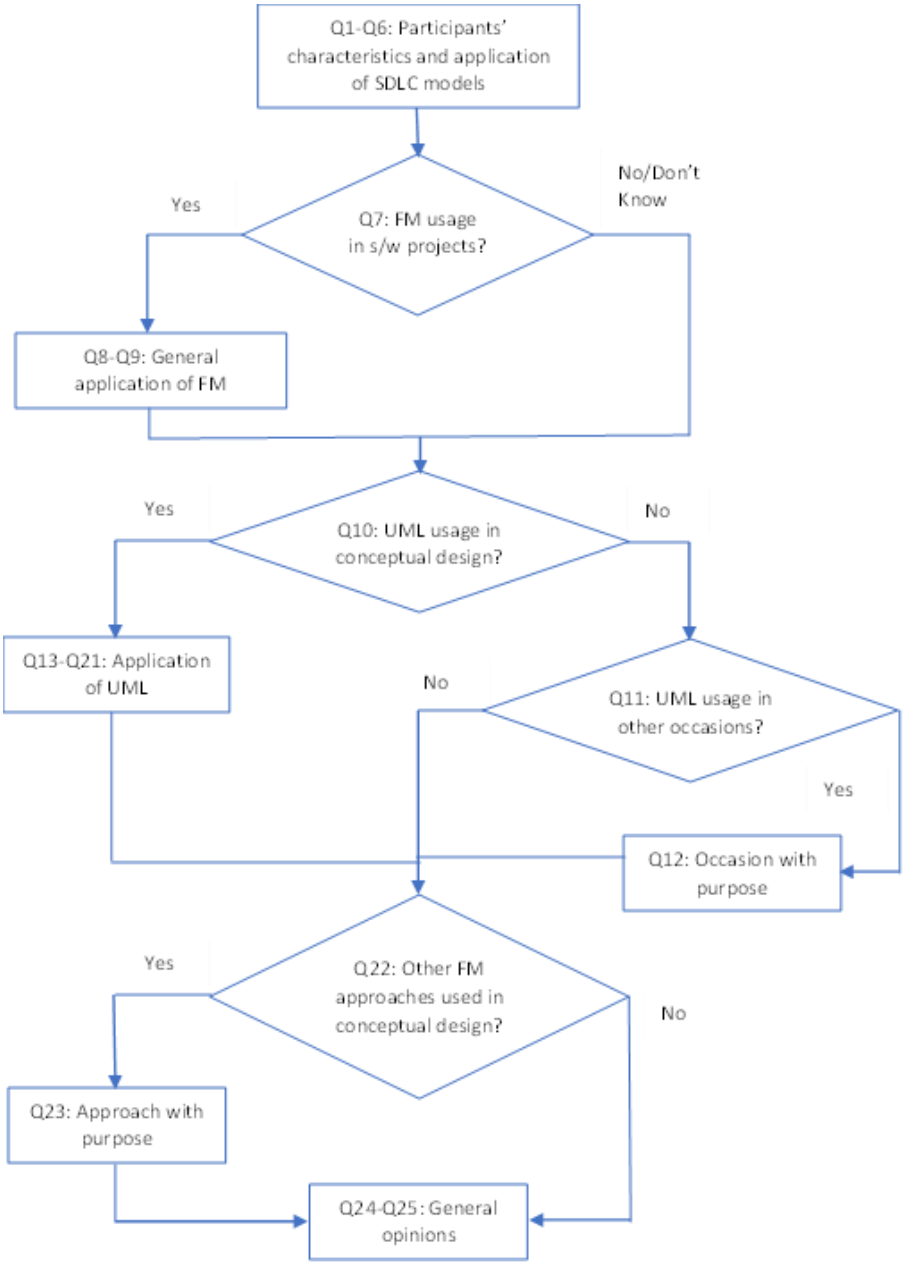


FIGURE 1: Structure of the Questionnaire.

3.2 Selection of Respondents

Respondents were contacted in 3 ways: (1) by sending emails to the students and researchers of the ISTD (Information Systems Technology and Design) pillar at SUTD, (2) by asking close contacts at two software development companies located in Singapore and Sri Lanka to distribute the questionnaire among their colleagues, and (3) by contacting friends and acquaintances who

work in the software industry via social media (e.g. LinkedIn). A total of 81 questionnaires were returned, 6 respondents did not fulfill the participation criterion of having at least 6 months of learning or working experience in software development.

3.3 Data Collection and Data Analysis

The questionnaire was distributed using Google Forms and collected, anonymously, into a separate Google sheet. Four open-ended questions were included in our survey. Three asked for specific function models and SDLC models. Only one question, Q23 on the purpose of using other function modelling approaches than UML, required a qualitative analysis. This question was coded and categorized using a Grounded Theory approach (Strauss & Corbin, 1997). The respondents' answers were clear and concise, allowing us to use their terminology to determine the codes. Our analysis essentially followed an inductive reasoning approach, as we did not have a strong hypothesis. The questions were based on assumptions derived from literature and experiences in software development.

4. RESULTS

This section starts with an analysis of the respondents' characteristics including their profession, experience, the number of involved projects, and their usage of UML (Section 4.1). In Sections 4.2-4.8 the research questions are addressed: application of function models in SDLC (Section 4.2), application of function models in conceptual design (Section 4.3), opinion on the importance of using UML in conceptual design (Section 4.4), purpose of using function models (Section 4.5), reasons for not using specific UML diagrams (Section 4.6), influence of the use of SDLC models on function model usage (Section 4.7), and the importance of function models in software development and their impact on the success of the software products (Section 4.8).

4.1 Respondents' Characteristics and Modelling Approaches

The respondents came from a variety of professions, which we categorized into:

- Software developers: developers and architects
- Analysts
- Other practitioners: software testers and project managers
- University students and researchers

This categorization allowed for a comparison with existing literature and for sufficient numbers per category.

Of the 75 respondents, 81.4% (n=61) are practitioners and 18.6% (n=14) are from academia. More than half of the respondents (52.0%, n=39) are software developers (see TABLE 1 for details).

Profession Category	Profession	# of Respondents	% of Respondents
Practice (n=61, 81.4%)			
Software developers	Software developer	39	52.0%
	Software architect	2	2.7%
Analysts	Analyst	14	18.7%
Other practitioners	Software tester	5	6.7%
	Project manager	1	1.3%
Academia (n=14, 18.6%)			
Academia	PhD student	5	6.7%
	Undergraduate student	4	5.3%
	Master's student	4	5.3%
	Researcher	1	1.3%
		75	**100.0%**

TABLE 1: Respondents' Profession (n=75).

Respondents' experience varies between 2 to more than 15 years with just under half of the respondents (45.4%) having 2-5 years of experience in software development (see TABLE 2). The majority (88.0%, n=66) have been involved in more than 3 software development projects, 52.0% (n=39) in more than 6 software projects, and half of these in more than 10 projects (see TABLE 3). Interestingly, even the users with a smaller number of years of experience in software development have participated in several software projects (see TABLE 4). For instance, of 24 respondents who have 1-2 years of total experience, 11 have been involved in 3-5 projects while 6 have been involved in more than six projects. Similarly, half of those who have 2-5 years of experience in software development has been involved in more than 6 software projects

Total Years of Experience	# of Respondents	% of Respondents
Practice (n=61)		
1-2	17	22.7%
2-5	29	38.7%
5-10	10	13.3%
10-15	2	2.7%
More than 15	3	4.0%
Academia (n=14)		
1-2	7	9.3%
2-5	5	6.7%
5-10	1	1.3%
10-15	1	1.3%
More than 15	-	-
	75	**100.0%**

TABLE 2: Respondents' Years of Experience in Software Discipline (n=75).

Number of Software Projects	# of Respondents	% of Respondents
Practice (n=61)		
1-2	6	8.0%
3-5	19	25.3%
6-10	19	25.3%
More than 10	17	22.7%
Academia (n=14)		
1-2	3	4.0%
3-5	8	10.7%
6-10	-	-
More than 10	3	4.0%
	75	**100.0%**

TABLE 3: Number of Software Projects the Respondents were Involved in (n=75).

Number of Projects / Total Experience	1-2 (n=9)	3-5 (n=27)	6-10 (n=19)	More than 10 (n=20)
1-2 Years (n=24)	7	11	4	2
2-5 Years (n=34)	2	15	9	8
5-10 Years (n=11)	-	1	4	6
10-15 Years (n=3)	-	-	1	2
More than 15 Years (n=3)	-	-	1	2

TABLE 4: Respondents' Total Years of Experience and Number of Software Projects Involved in (n=75).

The respondents were asked the following 3 questions on their usage of function models in SDLC and the usage of UML and other function modelling approaches specifically in the conceptual design phase (survey questions Q7, Q10, and Q22):

- *"Have you used any function modelling approach for projects you are/were involved with? If Yes, please mention them"*

- *"Are you currently using, or have you used UML FOR CONCEPTUAL DESIGN?"*
- *"Do you use any function modelling approach OTHER THAN UML for conceptual design?"*

Of the 75 respondents, 93.0% (n=70) answered that they are using function models in different phases of software development. 70.6% (n=53) of respondents stated that they use UML and 60% (n=45) that they use other function modelling approaches for conceptual design: 40% (n=30) are using both UML and other approaches.

The UML users were also asked about their experience (Q13 and Q14):
- *"Please indicate the number of years you have been practicing UML for conceptual design"*
- *"Please indicate the number of projects in which you used UML for conceptual design"*

Of the 53 UML users, 88.7% (n=48) have less than 5 years of UML experience, 81.2% (n=43) have participated in less than 5 UML projects, while only 11.3% (n=6) have been involved in more than 10 UML projects (see TABLE 5 and TABLE 6). As can be expected, users with fewer years of experience in UML also have been involved in fewer UML projects (see TABLE 7).

Years of UML Experience	# of Respondents	Percentage
Practice (n=42)		
1-2	21	39.6%
2-5	16	30.2%
5-10	5	9.4%
Academia (n=11)		
1-2	5	9.4%
2-5	5	9.4%
10-15	1	1.9%
	53	**100.0%**

TABLE 5: Respondent's Years of Experience in UML (n=53).

# of UML Projects	# of Respondents	Percentage
Practice (n=42)		
1-2	16	30.2%
3-5	18	34.0%
6-10	4	7.5%
More than 10	4	7.5%
Academia (n=11)		
1-2	8	15.1%
3-5	1	1.9%
More than 10	2	3.8%
	53	**100.0%**

TABLE 6: Number of UML Projects in which Respondents were Involved (n=53).

No of UML Projects / UML Experience	1-2 (n=24)	3-5 (n=19)	6-10 (n=4)	More than 10 (n=6)
1-2 Years (n=26)	16	8	2	-
2-5 Years (n=21)	7	11	2	1
5-10 Years (n=5)	1	-	-	4
10-15 Years (n=1)	-	-	-	1

TABLE 7: Comparison of Respondents' Years of UML Experience with the Number of UML Projects Involved in (n=53).

TABLE 8 shows the professions of these 53 UML users and the total number of respondents in each profession. The table shows that the majority of respondents use UML in software projects, except for software developers: 15 of 39 software developers did not use UML in their projects. Of these 39 developers, 12 have less than 5 years of experience in software development, and 8 have been involved in more than 6 software projects.

Profession Category	Profession	# of UML Users	% of UML Users		# of Total Respondents	# of Non-UML Users
Practice		**n= 42**			**n=61**	**n=19**
Software developers	Software developer	24	45.3%		39	15
	Software architect	2	3.8%		2	-
Analysts	Analyst	12	22.6%		14	2
Other practitioners	Software tester	3	5.7%		5	2
	Project manager	1	1.9%		1	-
Academia		**n=11**			**n=14**	**n=3**
Academia	PhD student	4	7.5%		5	1
	Undergraduate student	3	5.7%		4	1
	Master's student	3	5.7%		4	1
	Researcher	1	1.9%		1	-
		53	**100.0%**		**75**	**22**

TABLE 8: Professions of the UML Users.

4.2 RQ1: Application of Function Models in Software Design

The answer to research question 1: *Which function models are used for software design? (personally, as well as within the organization)* was obtained by analyzing the answers to survey questions Q8 and Q9 in which respondents were asked for the function modelling approaches they use personally or are used by their organization.

For personal use, answers from only 65 respondents were analyzed, as 5 of the 70 who use function models did not mention any function modelling approach. From those 65, 128 responses were obtained as multiple answers were allowed. The results (see TABLE 9) confirm that UML has the highest personal usage among all approaches with 75.4% (n=49). This is followed by flow charts (n=31), DFD (n=19), block diagrams (n=12), and state diagrams (n=11). It is clear that structured function models like DFD, ERD, and state diagrams, are still used, at least to some extent, with DFD as the most popular with 29.2%.

Function Modelling Approach	# of Respondents	Percentage
UML	49	75.4%
Flow chart	31	47.7%
Data Flow Diagram (DFD)	19	29.2%
Block diagram	12	18.5%
State diagrams	11	16.9%
Entity Relationship Diagram (ERD)	3	4.6%
Process diagrams	2	3.1%
Architecture diagrams	1	1.5%

TABLE 9: Function Modelling Approaches Used by the Participants in Software Projects (n=65).

When asked about any *other* function modelling approaches that their organizations use frequently, only 14 responses were received from 10 respondents: 9 practitioners (2 analysts, 6 software developers, and 1 software architect), and one master's student. The two analysts mentioned UML activity and sequence diagrams, while the software developers and architects mentioned block diagrams, DFDs, state chart diagrams. Flow charts were mentioned by both the practitioners and the master's student. Similar to personal use (see TABLE 9), UML, flow charts,

and block diagrams are scored highest with 3 responses each. Only one respondent, a software developer, mentioned user stories and storyboards. Although these findings confirm the individual practices, the number of responses does not allow us to draw conclusions at the organizational level. Possible reasons for the low number of responses are that the organization uses the same function modelling approach as the participants, or the participants do not know whether or which other function modelling approaches are used in the organization.

4.3 RQ2: Application of Function Models in Conceptual Design

This section presents the usages of function models specifically in the conceptual design stage, to answer research question 2: *How are these function models applied during conceptual design?* The answers were analyzed in two separate categories: UML and other function modelling approaches.

4.3.1 Usage of UML in Conceptual Design

The analysis considered all the diagrams in UML 2.5 except for the profile diagram. Use case narratives are considered a part of UML as these describe use case diagrams (El-Attar & Miller, 2006).

Respondents who used UML (n=53) were shown a list of the resulting 14 UML diagrams and were asked to indicate for each diagram whether it was '*frequently used', 'used', 'scarcely used', or 'never used*' (Q17). FIGURE 2 shows three groups based on the total of "frequently used" and "used".

1. Sequence, use case, class, and activity diagrams are clearly the **most used**, as a large number (>71%) of respondents either frequently or normally used these diagrams. Of these, the sequence diagram was most often used (84.9%), followed by use case, class, and activity diagrams with 77.4%, 75.5%, and 71.7%, respectively.
2. Component, deployment, object, and state chart diagrams are **moderately used,** i.e. between 47% -51% of respondents used or frequently used these diagrams.
3. The remaining diagrams - interaction overview (IOD), use case narrative, package, communication, timing, and composite structure (CSD) - are clearly **used less**.

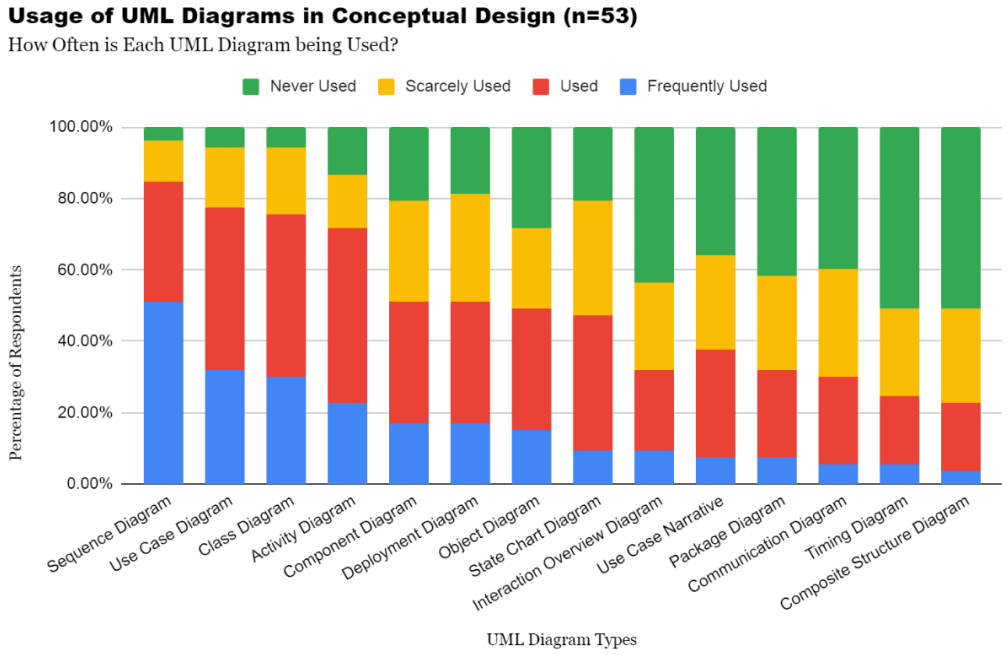


FIGURE 2: Percentages of Respondents and Usage of Different UML Diagrams in Conceptual Design (n=53).

Even though UML contains many diagrams and – as just shown - all of them were used by at least some of the respondents, the results show that respondents only use a handful of diagrams frequently. It is interesting to note that for each UML diagram some percentage (3.8%- 51.0%) of respondents (of 53) '*Never used'* these (see FIGURE 2). The diagrams that were used less often, were also those that were most often mentioned to be *'Never used'*. Every professional group had respondents who '*Never used'* at least one diagram – 15 (of 26) are software developers, 7 (of 12) are analysts, 2 (of 4) are other practitioners and 8 (of 11) are users from academia. Apart from 6 practitioners, all had ≤5 years of experience in software development.

Another noticeable finding is that the four diagrams which are most frequently used in the conceptual design stage - sequence, use case, class, and activity diagrams - represent structural and behavioral aspects of the system rather than architectural aspects. This confirms the findings of Ozkaya & Erata (Ozkaya & Erata, 2020) as they also found practitioners often choose sequence and activity diagrams for data flow modelling, which is a behavioral aspect, whereas the class diagram is the top choice for functional structure and data structure modelling. Yet, surprisingly, use case narratives was far less used compared to use case diagrams. Dobing and Parsons (Dobing & Parsons, 2008) observed a similar result in their survey among 284 software analysts. They too found low usage of use case narratives, and one-third mentioning that they never used it. A possible explanation is that users prefer diagrams over text when describing scenarios. Our findings also confirm Dobing and Parson's results that most users prefer sequence diagrams over communication diagrams, even though these diagrams have very similar representations.

The low usage of timing diagrams may be explained by the fact that UML offers simpler diagrams such as sequence and state charts, to represent the same product features. This may also account for the low use of IOD, which is a complex combination of widely used diagrams like sequence and activity diagrams. A reason for the low usage of CSD and package diagrams may be that people do not yet think about the architectural components during conceptual design. A further possible reason for low use may be the lack of knowledge and familiarity with these diagrams as suggested by both Dobing and Parsons (Dobing & Parsons, 2008) and our own study (see Section 4.6).

TABLE 10 shows a comparison of our results on UML diagram usage with those of other empirical studies (Dobing & Parsons, 2008; Erickson & Siau, 2007; Reggio et al., 2014; Reggio et al., 2013; Wrycza & Marcinkowski, 2007). Our comparison is based on the terminology the authors used in their publications. It is important to note that the definitions of 'widely used' vary: Reggio et al. (Reggio et al., 2014) refer to the number of people using a diagram, whereas the other authors and our study refer to how frequently people use a diagram. Another difficulty is that we do not know the Likert scale options used by Reggio et al. (Reggio et al., 2014), Wrycza and Marcinkowski (Wrycza & Marcinkowski, 2007), and Erickson and Siau (Erickson & Siau, 2007), and hence do not know which options were combined in their conclusions.

Reference and Year of Data Collection	**Source**	**Basic Analysis/ Categorization**	**Main Results**
Our study (2020)	75 respondents (61 practitioners, 14 from academia)	• Respondents were asked how frequently they used the diagrams: Frequently used, used, scarcely used, or not used	• Most Used (>71%): sequence, use case, class, activity diagrams • Moderately used (51%-47%): component, deployment, object, state chart diagrams • Used less (<37%): use case narrative, IOD, package, communication, timing, CSD diagrams

(Reggio et al., 2014) (study in 2013-2014)	275 respondents (60% practitioners and 40% from academia)	• Respondents were asked whether they used each diagram and based on the answer percentage the diagrams were categorized into 'widely used', 'used', and 'scarcely used'	• Widely used (used by 80%- 67% of respondents): sequence, class, use case, state chart, and activity diagrams • Used (48%-32%): package, component, object, deployment, communication diagrams • Scarcely used (21%-12%): CSD, IOD, and timing diagrams
(Wrycza & Marcinkowski, 2007) (study in 2007)	180 university students	• Results based on the percentage of respondents who used the diagrams 'mostly'	• Most used (62%-21%): class, use case, activity, sequence diagrams • The aim was to find a light version of UML; hence these 4 diagrams are considered as the light version of UML 2.0
(Dobing & Parsons, 2008) (study in 2003-2004)	284 analysts	• Respondents were asked about the proportion of the UML projects that they have been involved with have used the given 7 UML diagrams • A five-point scale was used: None, 1/3, 1/3-2/3, >2/3, and All	• The ranking of diagram usage in two-thirds or more of their project is class (73%), use case (51%), sequence (50%), use case narrative (44%), activity (32%), state chart (29%), and collaboration (22%)
(Erickson & Siau, 2007) (study in 2003-2004)	29 participants (13 academics, 9 students, and 7 practitioners from the computer and finance industry)	• Participants were asked to rate the diagrams on a 0-5 scale with 0 being never used. • Mean and standard deviations were calculated to find the most used diagrams.	• Most used: class, use case, sequence, and state chart diagrams (>90%) • Class, use case and sequence, diagrams considered the core diagrams
Presence in Literature			
.(Reggio et al., 2013) (study in 2013)	UML books - 30 Tools – 20 Courses – 22 Tutorials -18	• Widely used and scarcely used if the diagram is present in >60% and <40% of the sources, respectively • 60%-40%: non defined cases (grey zone)	• Widely used: class, activity, sequence, state chart, use cases, communication, deployment, component, object, package diagrams • Grey zone: CSD • Scarcely used: timing, IOD, and profile diagrams

TABLE 10: Summary of the Empirical Studies Found in the Literature on the Use of UML Diagrams.

TABLE 11 allows a comparison between the rank order and categorization of UML diagram usage resulting from our study with that of the other studies listed in TABLE 10. The empirical studies are placed in reverse chronological order. The row *'participants'* indicates the relative number of participants: academia, practice, or students (see TABLE 10 for the actual numbers). Until 2005, UML only included 8 diagrams (UML V1.X). After that, 6 more diagrams were introduced with UML V2.0. These are marked with asterix (*) an in TABLE 11.

Overall, the results of the various studies are quite similar, but we found some notable differences. Whereas sequence, use case and class diagrams remain the most used diagrams, the state chart diagram ranked 4th in three of the studies, but only 8th in our study and 13th (last) in Wrycza and Marcinkoswki's 2007 study. The latter also ranked deployment diagrams much

lower. A possible reason for the discrepancy can be their respondents: they only involved students.

	Our Study	**Reggio et al. (Reggio et al., 2014)**	**Wrycza and Marcinkowski (Wrycza & Marcinkowski, 2007)**	**Dobing and Parsons (Dobing & Parsons, 2008)**	**Erickson and Siau (Erickson & Siau, 2007)**	**Reggio et al. (Reggio et al., 2013)**
Year of study	2020	2013-2014	2007	2003-2004	2003-2004	<2013
Participants	pract>acad	pract>acad	stud	pract	acad>pract	Educational material
Sequence	1	1	4	3	3	3
Use Case	2	3	2	2	2	4
Class	3	2	1	1	1	1
Activity	4	5	3	5	6	2
Component	5	7	7	-	5	6
Deployment	6	9	11	-	8	6
Object*	7	8	5	-	9	7
State Chart	8	4	13	6	4	4
Use Case Narrative	9	-	-	4	-	-
Package*	10	6	9	-	-	8
IOD*	11	12	8	-	-	11
Communication /Collaboration	12	10	6	7	7	5
Timing*	13	13	12	-	-	10
CSD*	14	11	10	-	-	9
Profile*	-	11	-	-	-	12

TABLE 11: Rank Comparison of UML Diagram Usage (1 = Most Used).

	Ranks 1 – 4		Not included in the respective study
	Ranks 5 – 8	*	Only approved by Object Management Group (OMG) in 2005 as part of UML2.0
	Ranks 9 – 14		

Our study and that of Dobing and Parsons (software analysts only) are the only ones to include use case narratives. Use case narratives ranked 4th in Dobing and Parsons's study whereas it ranked 9th in our study. However, the percentage of usage is almost the same in both studies as it has been used by 44% in their study and 37.7% in our study. Dobing and Parsons mentioned that this usage is below their expectations as UML is developed based on the use case-driven philosophy. We identified package and communication diagrams as being **used less** often with <30% of respondents selecting options '*frequently used'* and *'used'*. The study of Reggio et al. (Reggio et al., 2013) suggest that this may not be because of a lack of familiarity. They found that these two diagrams are among the most often included in books, tools, and courses on UML. Although the order of the diagrams shown in FIGURE 2 is almost the same as the results of Reggio et al.'s (Reggio et al., 2014) study on the actual use of diagrams, our overall percentages are a bit higher. Note that Reggio et al. refer to the number of people whereas we refer to the frequency of use.

Compared to Wrycza and Marcinkowski (Wrycza & Marcinkowski, 2007), their most-used diagrams are the same as our **most frequently used** ones, but the highest use frequency was 62% while we found 85% for sequence diagrams and 71.7% for activity diagrams. Interestingly, in their study state chart diagrams were found to be the least used with <10%, whereas we found 48%. A possible reason for this difference may again be their respondents, who were university

students, whereas our respondents were mainly practitioners. As shown in FIGURE 3, however, 63.6% of our respondents from academia also used state chart diagrams, similar to the practitioners. This is more in line with Erickson and Siau (Erickson & Siau, 2007), 100% of their mainly academic (76%) participants used state chart diagrams. The differences may be due to what has been taught in the involved universities but also suggests a discrepancy between education and practice.

The analysis did not show a clear relationship between the usage ('*frequently used'* or '*used*') of diagrams and the respondents' professions (see FIGURE 3), but we see some tendencies. Compared to analysts, software developers and other practitioners more often use structural and implementation diagrams, such as class, object, package, component, and deployment. Respondents from academia and analysts seem to prefer 4 diagrams – class, sequence, use case, and activity diagrams, closely followed by state chart diagrams only for respondents from academia. The other practitioners (3 software testers and 1 project manager) use structural diagrams more often than the other participants, which could be due to their profession, i.e. testing. Their number, however, is too small to draw any conclusions.

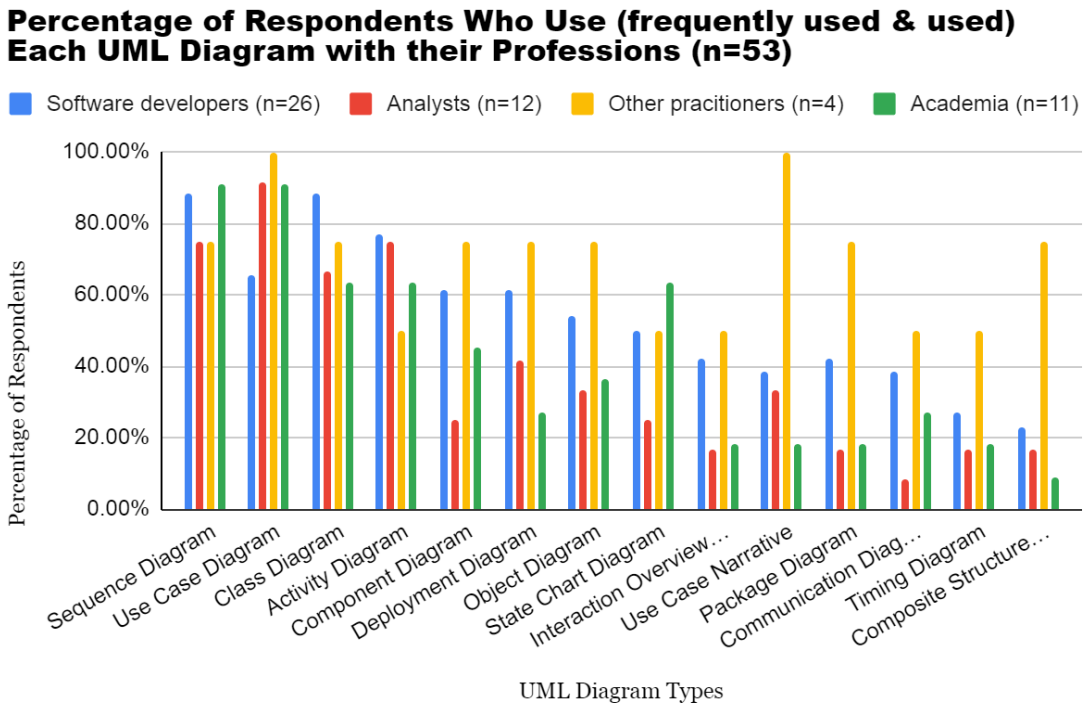


FIGURE 3: UML Diagram Usage ('*frequently used*' or '*used*') with Respondents' Profession (n=53).

4.3.2 Selective Use of UML

According to Jacobson, "*20% of language is needed to develop 80% of software*" (Grossman et al., 2005; Reggio et al., 2013). Based on this so-called Pareto principle, we could expect most users to use only a few of the 14 models provided by UML. Petre (Petre, 2013) claims that UML is useful when it is used selectively, otherwise, it leads to an increase in the complexity and inconsistency of the design.

We, therefore, asked our respondents about the average number of UML diagrams that they used per project (Q19). Of 53 UML users, 37 responded. The analysis showed an average of 3.4 diagrams, with a minimum of 1 and a maximum of 8 diagrams. Just over half (51.4%) used 3 diagrams per project (see FIGURE 4), which is in line with the Pareto principle. No correlation was found between the number of diagrams used with any of the respondent's characteristics.

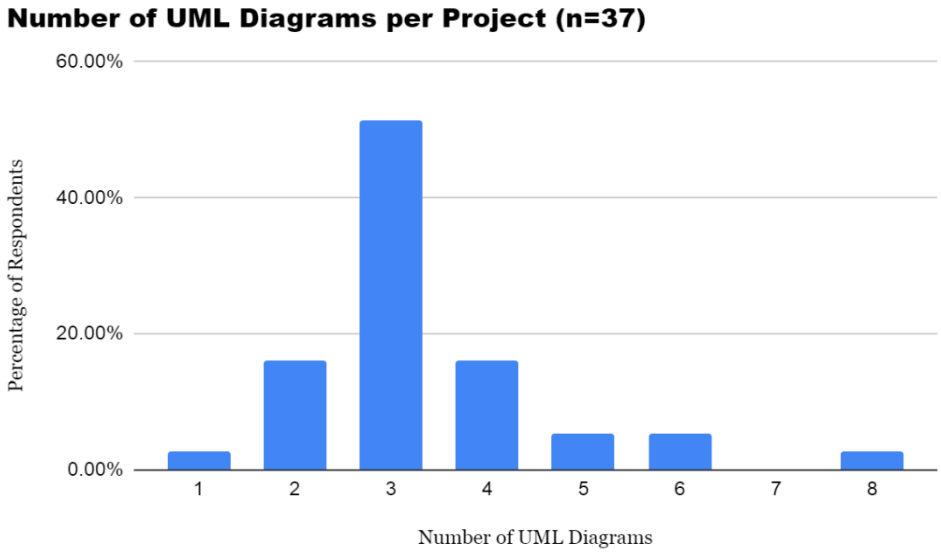


FIGURE 4: Number of UML Diagrams (of 14 Available Diagrams) per Project (n=37).

4.3.3 UML Usage Patterns

The respondents were asked about the usage of UML in their organizations during conceptual design (Q16). As shown in TABLE 12, only a quarter of the respondents mention that UML is used consistently in all development projects, and another one-third mentions that UML is used only for larger projects. This is an increase compared to the study of Grossman et al. (Grossman et al., 2005) more than 15 years ago. At the time almost half of the respondents stated that they used UML sporadically. They argued that this is because UML was new and not adopted yet by many companies. Our study suggests that companies still are not using UML for all of their projects. A possible reason may be a lack of encouragement, support, and resources provided by the management to the software teams, as suggested by Grossman et al. (Grossman et al., 2005) who found that >60% of management did not encourage development teams to use UML or did not provide necessary resources. Moreover, their findings suggest that many organizations do not seem to care about methods used as long as a project is completed on time.

UML Usage	# of Respondents	Percentage
Used consistently in all development projects	14	26.4%
Used only for large projects	18	34.0%
Used sporadically	16	30.2%
I don't know	5	9.4%
	53	**100.0%**

TABLE 12: UML Usage in Organizations (n=53).

4.3.4 Usage of Other Function Modelling Approaches in Conceptual Design

This section presents the use of function modelling approaches other than UML in conceptual design (Q23). Of the total of 75 respondents, 60% (n=45) stated that they use function modelling approaches other than UML during the conceptual design stage. As 6 respondents did not mention any function modelling approach, only the responses of 39 respondents were analyzed. Some respondents mentioned multiple approaches to this open-ended question, resulting in 55 responses in total.

User stories were the most often used non-UML function modelling approaches with 59% of respondents (see TABLE 13). A possible reason for this high rate may be the growth of agile

programming practices: user stories are part of agile programming. As already suggested by Ambler (Ambler, 2004) in (Dobing & Parsons, 2008), UML has to update to support this growth. Our results further highlight that structured models, such as flow charts, ERD, and DFD are still used for conceptual design, but by far fewer people (25.6%-18%). As discussed in Section 4.1, of 75 survey respondents, 40% (n=30) mentioned that they use both UML and other function models, suggesting that one function modelling approach is not sufficient to design software.

Function Modelling Approach	# of Respondents	Percentage
User stories	23	59.0%
Flow charts	10	25.6%
Entity Relationship Diagram (ERD)	9	23.1%
Data Flow Diagram (DFD)	7	18.0%
Block diagram	4	10.3%
Business Requirement Document (BRD)	1	2.6%
Unified approach	1	2.6%

TABLE 13: Non-UML Function Modelling Approaches Used for Conceptual Design (n=39).

FIGURE 5 shows some dependencies between respondents' professions and the non-UML function models. User stories are the non-UML diagrams most used by other practitioners (100.0%), analysts (90.0%), and those in academia (66.7%). DFD's are popular with other practitioners (66.7%) and to a lesser extent in academia (33.3%). Software developers do not seem to use non-UML modelling approaches very frequently (<35% for each approach). If they do, these are ERD, user stories, flow charts, and block diagrams. BRD is only used by one participant, an analyst. The unified approach and block diagram are only used by very few software developers.

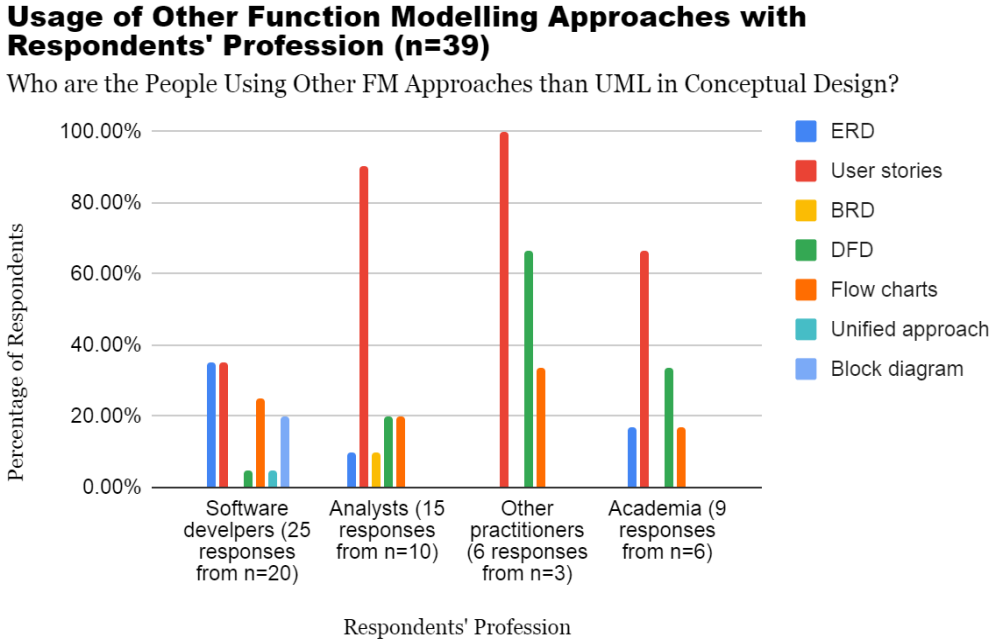


FIGURE 5: Non-UML Function Modelling Approaches Used with Respondent's Profession (n=39).

In summary, UML, which according to the literature is the most beneficial function modelling approach (Ozkaya, 2019), was also found to be the most widely used function modelling

approach among the respondents of our study. However, only a few of the 14 UML function models are popular and per project, only 2-4 models are used. At the same time, traditional structured and data-driven function models are still in use, such as DFD, ERD, and flow charts, but less frequently. User stories are a popular non-UML approach in academia and with analysts and other practitioners, but not so much for software developers. In general, software developers tend not to use non-UML approaches.

4.4 RQ3: Opinion on the Importance of UML in Conceptual Design

This section addresses research question 3: *How important are the function models in the conceptual design phase?* by analyzing the answers for survey question Q18 in which the UML-using respondents (n=53) were asked to provide their opinion on the importance of each of the 14 UML diagrams in conceptual design, irrespective of whether they had applied the diagram or not. We used a 5-point Likert scale with options: '*very important', 'important', 'slightly important', 'not at all important'* for conceptual design, and *'no opinion*'. FIGURE 6 shows large differences between the diagrams. Based on the percentage of users selecting either '*very important'* or '*important'*, the diagrams were categorized as ***Most Important, Moderately Important, and Least Important.***

1. ***Most Important –*** Diagrams that are considered '*very important' or 'important'* for conceptual design by >80% of respondents are: use case, sequence, class, and activity diagrams.
2. ***Moderately Important –*** Diagrams that are considered '*very important' or 'important'* for conceptual design by >50% of respondents are deployment, state chart, object, communication, component diagrams, and use case narrative.
3. ***Least Important –*** Diagrams that are considered '*very important' or 'important* for conceptual design by <50% of respondents are: timing, package, interaction overview (IOD), and composite structure (CSD) diagrams.

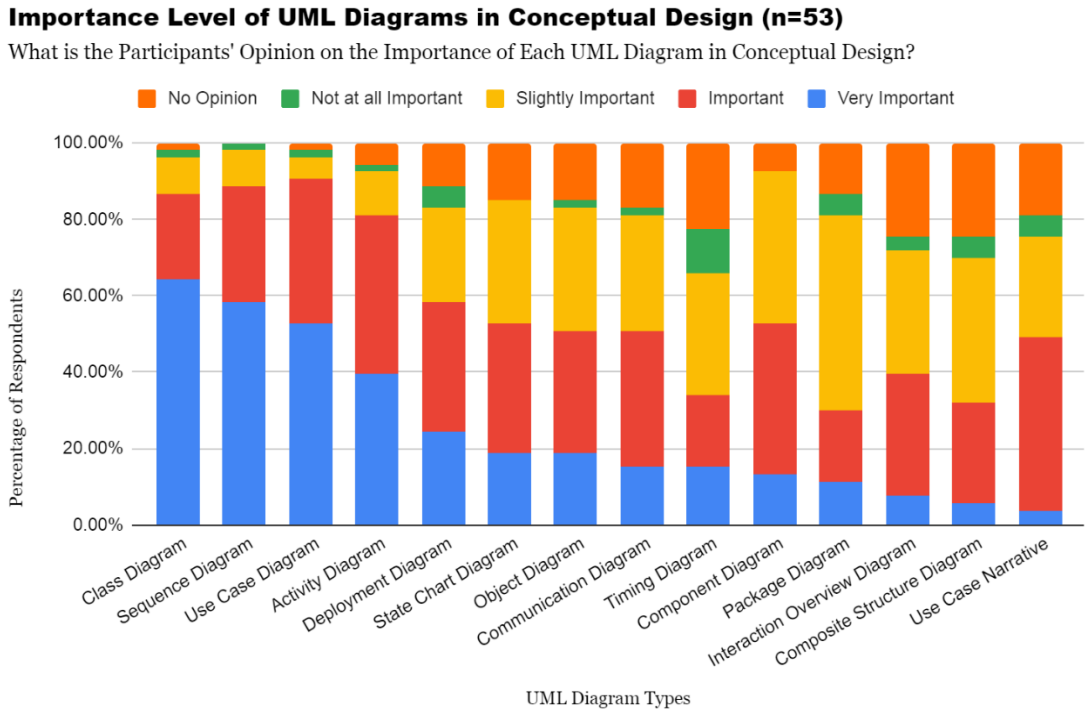


FIGURE 6: Opinion on Importance of Different UML Diagrams in Conceptual Design (n=53).

Class, sequence, and use case diagrams are highlighted as the most important diagrams as >52.0% rated these as '*very important'*, closely followed by activity diagrams with 39.6%. None of

the respondents considered state charts and component diagrams '*not at all important'* but some had '*no opinion'* about these. For all other diagrams, between 1.9%-11.3% (n=1-6) selected '*not at all important'*: the highest value (11.3%) is for the timing diagram. Interestingly, the sequence diagram was the only diagram about which each respondent had an opinion, which suggests that all respondents are familiar or at least aware of this diagram. For the other diagrams, 1.9%-24.5% of UML users (n=1-13) had no opinion, with timing, CSD, IOD, use case narrative, and communication diagrams scoring highest (>17%, n ≥9). This is in line with the findings described in Section 4.3.1, that >40% of respondents '*Never used'* timing, CSD, and IOD diagrams. A likely reason is the lack of familiarity with these diagrams (see Section 4.6).

Of the 22 respondents who selected '*no opinion'* or '*not at all important'* for a diagram in the conceptual design stage, 15 are practitioners of which the majority are software developers (n=9). Furthermore, of 17 who selected '*no opinion'* for at least one diagram, 14 have ≤5 years of experience in software development, while of 11 who selected '*not at all important'*, 9 have ≤5 years of experience. Irrespective of the profession, almost all of these 22 respondents think the least used diagrams: CSD, IOD, and timing are not at all important or they have no opinion. Interestingly, 6 respondents including analysts, a developer, an architect, a tester, and an undergraduate student considered at least one of the frequently used diagrams - class, activity, sequence, and use case – '*not at all important'* for conceptual design or had '*no opinion'*.

The importance levels ('*very important*' & '*important'*) correspond largely to the actual UML usage levels ('*frequently used'*, and '*used'*) described in Section 4.3.1 (Pearson correlation coefficient 0.96). As shown in FIGURE 7, in general, more respondents considered a diagram important than actually used it (average difference of 7.7%). The difference is very low (<3.7%) for sequence, object, and component diagrams, and largest for the communication, class, and use case diagrams and use case narratives (11.3%-20.8%).

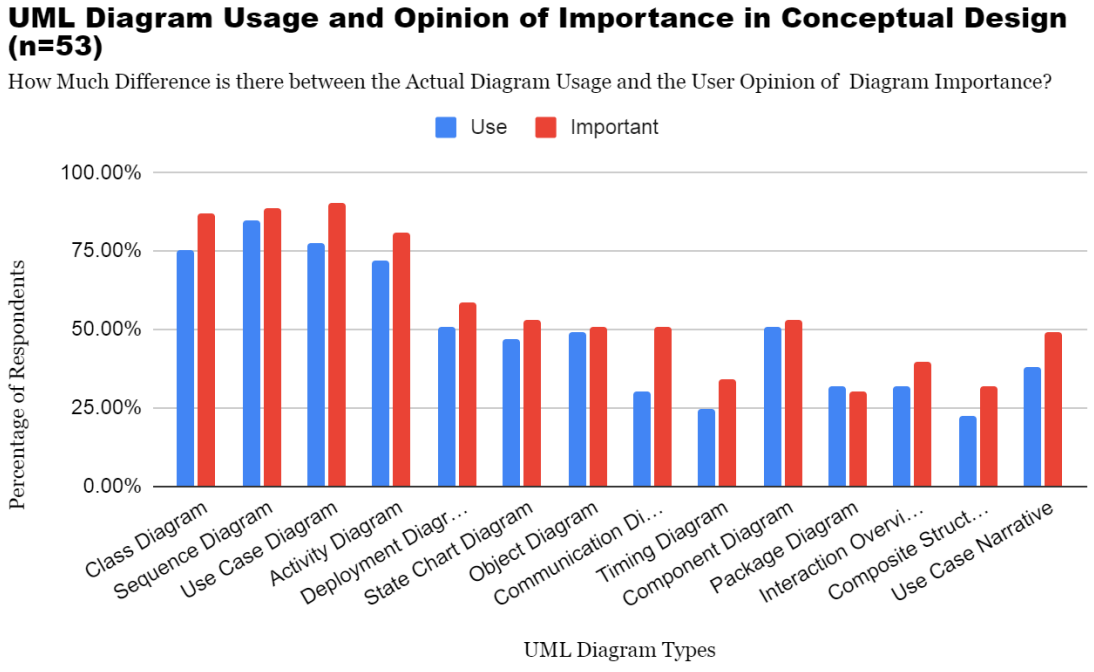


FIGURE 7: Comparison of Actual UML Diagram Usage ('*frequently used'*, and '*used'*) and Opinion of Importance ('*very important'* & '*important*') in Conceptual Design (n=53).

As shown in FIGURE 8, more than 75% of each professional category believes that the 4 most used diagrams – class, sequence, use case, activity diagrams – are '*very important*' or '*important'*

in conceptual design. Except for 'other practitioners', all respondents believe that the diagrams we categorized as **used less** are not important in conceptual design.

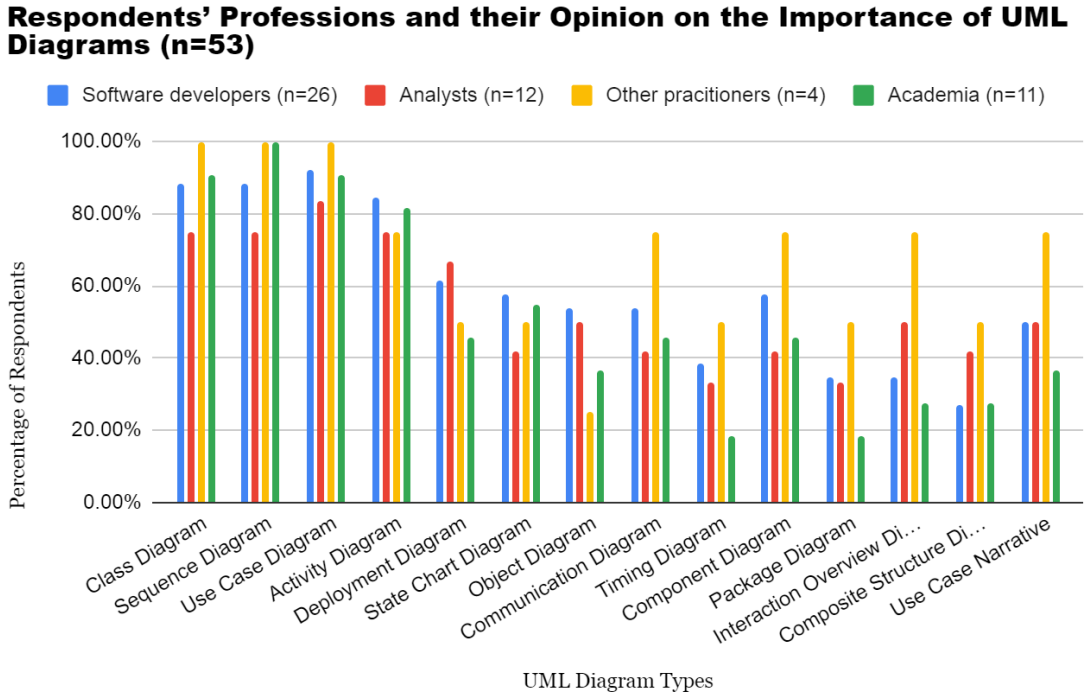


FIGURE 8: Percentage of Respondents of a Particular Profession who Consider a Particular UML Diagram *'very important'* or '*important'* (n=53).

4.5 RQ4: Purpose of Using Function Models in Software Projects

In this section, we answer research question 4: *What are the reasons for using function models in software projects?* As in Section 4.3, the results are presented separately for UML and other function modeling approaches.

4.5.1 Purpose of Using UML Diagrams

The respondents were asked to select one or more reasons, shown in the top row of TABLE 14, for why they '*frequently used*' or '*used*' a particular UML diagram (Q20). The number of responses differs for each diagram, as the respondents were only asked to respond in case, they used the diagram. The interactional and behavioral diagrams have been highlighted in blue. For each diagram, the purposes that are mentioned by more than 50% of respondents have been highlighted in grey. The last column contains the average number of reasons given by the respondents who provided a reason for the use of a particular diagram.

On average 41.2% of the 53 UML users provided a reason for a diagram. Only use case, class, and sequence diagrams received responses from more than 35 of the 53 UML users (>67%), the other diagrams received responses from fewer than 27 respondents (<50%). Some respondents did not select any reason, even though they used the respective diagram (see first column).

n= Number of Respondents/ Actual Users	Verify and Validate Requirements with Clients	Specify Requirements for Programmers	Collaboration between Technical Teams	Generate Codes	Documenting for Future Upgrades	Other	Total Responses	Average per person.
Sequence (n=38/45)	24	22	14	1	12	-	73	1.9
Use Case (n=42/42)	36	16	8	-	11	-	71	1.7
Class (n=36/40)	7	21	18	9	9	-	64	1.8
Activity (n=27/38)	15	16	13	1	7	1	53	2.0
Component (n=19/19)	3	10	8	4	3	1	29	1.5
Deployment (n=18/27)	5	12	7	2	4	1	31	1.7
Object (n=23/26)	3	12	11	5	2	1	34	1.5
State Chart (n=15/25)	3	7	8	2	2	1	23	1.5
IOD (n=10/17)	2	5	3	-	1	2	13	1.3
Use Case Narrative (n=19/20)	15	9	3	1	3	1	32	1.7
Package (n=18/18)	2	8	7	2	1	1	21	1.2
Communication (n=15/16)	2	7	7	-	2	2	20	1.3
Timing (n=14/14)	3	6	7	1	1	2	20	1.4
CSD (n=12/12)	-	8	2	1	1	3	15	1.3

TABLE 14: Purpose of Using Each UML Diagram in Software Projects (Multiple Purposes Possible) (Blue = Interaction and Behavioral Diagrams, Grey = More than 50% of Respondents).

Verify and validate requirements with clients: Sequence, use case, and activity diagrams, as well as use case narratives, are widely used to verify and validate requirements with clients (each with >55% of responses). It is the main purpose of sequence, use case diagrams and use case narratives, and second for activity diagrams. The other diagrams are barely used for this purpose.

Specify requirements for programmers: This is the purpose of many of the diagrams: for 9 of the 14 diagrams this purpose was mentioned by over 50% of respondents, for the remaining diagrams by over 38% of respondents, with use case diagrams being the lowest.

Collaboration between technical teams: Except for use case diagrams, use case narratives, and CSD, all other diagrams were considered to be moderately useful for collaboration between technical teams (mentioned by > 30% of respondents). It is the main purpose (with over 50%) for state chart and timing diagrams, and the second most important purpose of class diagrams. As for '*specify requirements for programmers'*, sequence, class, and activity diagrams show the highest values, while IOD, use case narrative and CSD are hardly used for this purpose.

Generate codes: Very few respondents use UML for code generation. Class, component, and object diagrams are used for this purpose, but only by 25%-21% of its users.

Documenting for future upgrades and maintenances: Few respondents selected this purpose. Sequence, use case, class, activity, and deployment diagrams are used for this purpose, but only by 32%-22% of its users.

Other reasons: Only very few respondents indicated that they used a diagram for other reasons. Those respondents who '*scarcely used*' a particular diagram, in general, used it for the same purposes as those who '*frequently used*' or '*used'* the diagram. The numbers are too low, however, to draw any conclusions.

The results highlight that the main use of the UML diagrams is to communicate requirements with clients and programmers and – to a lesser extent – to collaborate between technical teams. The diagrams used for the communication with clients are interaction and behavioral diagrams (marked blue in TABLE 14), whereas, for communication with programmers and collaboration between teams, interaction and behavioral as well as structural diagrams are used. This confirms the findings of Dobing and Parsons (Dobing & Parsons, 2008): use case narratives, activity, and use case diagrams are mainly used for working with clients, whereas class diagrams when working with technical team members. Use case diagrams and narratives seem to be particularly useful in communications with the clients (>78% of respondents). The reason may be that they represent high-level scenarios of the system, which are easy to understand. As mentioned before, UML is not used often to generate codes. This concurs well with the findings of Wrycza & Marcinkowski (Wrycza & Marcinkowski, 2007) and those of Petre (Petre, 2013) who mentioned that codes generated via UML are not production-ready. Documenting for future purposes is not considered an important purpose for most diagrams: the most often mentioned (31%-25% of respondents) were used sequence, use case, and class diagrams. A reason may be that people tend to continue to use the diagrams they frequently use (see Section 4.3.1) rather than changing to a different one just for documentation purposes.

Looking at the professions, **analysts** widely use UML for two main reasons: verify requirements with clients and specify requirements for programmers, which reflects their key roles in practice. They use sequence, use case, activity, state chart diagrams, and use case narratives mainly for client communication. To communicate with programmers there does not seem to be a preference: almost all the diagrams are used moderately for this purpose. This supports the finding of Aranda (Aranda, 2010) that UML is mainly used by analysts for business communications. Our study, however, does not confirm his finding that function diagrams are ignored by other professions: **developers, architects, and other professions such as testers** do use function diagrams. In fact, they stated that they utilize UML diagrams mainly to specify the requirements and collaborate with the technical teams and that they utilize almost every UML diagram for these 2 purposes, except for use case diagrams and narratives. Some also utilize sequence diagrams, use case diagrams, and use case narratives to communicate with clients, and utilize class, object, component, and state chart diagrams to generate codes. For **people from academia**, the key reason for using UML appears to be the collaboration between teams, rather than with clients, which can be expected in an academic context. They widely use class, activity, communication, state chart, object, component, and timing diagrams for this purpose. When the respondents from academia communicated requirements with clients, they also use sequence diagrams, use case diagrams and use case narratives.

4.5.2 Purpose of Using Other Function Modelling Approaches in Conceptual Design

When asked about the use (see Section 4.3.4) and purpose of other, non-UML function modelling approaches in conceptual design, we received 55 answers from 39 respondents. The question was open-ended, in contrast to the question about the purpose of UML approaches. The answers were categorized as shown in TABLE 15, closely following the – generally very clear - terminology of the respondents.

Non-UML function modelling approaches are not so widely used (as shown earlier in TABLE 13), but when they are used their purpose is similar to UML diagrams (see TABLE 14). The most often mentioned purposes (TABLE 15) are: to capture or verify & validate the requirements from clients (53.8%, n=21), to communicate requirements with programmers (25.6%, n=10), and to understand the requirements or system functionality (17.9%, n=7). Other reasons received very few responses (≤10.3% each).

N= Number of Users n= Number of Respondents	% of 39 Respondents	User Stories (N=23)	Flow Charts (N=10)	ERD (N=9)	DFD (N=7)	Block Diagram (N=4)	BRD (N=1)	Unified Approach (N=1)
Capture, verify & validate, clarify and obtain approval for requirements from clients (n=21)	53.8%	14	3	-	2	1	1	-
Specify requirements for programmers (n=10)	25.6%	5	-	4	-	-	-	1
Understand the requirements, system functionality (n=7)	17.9%	-	3	-	2	2	-	-
Collaborate between technical teams (n=4)	10.3%	4	-	-	-	-	-	-
Document for upgrades and maintenance (n=4)	10.3%	-	1	2	1	-	-	-
Describe internal flows and structures of the system (n=4)	10.3%	-	2	1	1	-	-	-
Understand the system design and structure (n=3)	7.7%	-	1	1	1	-	-	-
Generate codes (n=2)	5.1%	-	-	1		1	-	-

TABLE 15: Purpose of Using Non - UML Function Modelling Approaches in Conceptual Design (n=39).

User stories are by far the most used non-UML approach. Dalpiaz & Brinkkemper (Dalpiaz & Brinkkemper, 2018) claim that 90% of agile practitioners use user stories to capture requirements and it has become much popular because of its simple structure. Similar to the literature, we also observed that user stories are mainly used to clarify or verify & validate the requirements with clients and specify these for programmers. Interestingly, user stories are the only approach that was mentioned for the collaboration between technical teams. Flow charts, and to some extent ERD and DFD, tend to be used for different purposes than user stories. Their purposes are related to the system's functionality and documentation.

The results for UML and other approaches were analyzed separately as we expected some differences in purposes. We found, however, that the purposes are largely the same: to support requirement-related tasks and communication. Other purposes are also mentioned, but those models specific, related to the specific features and benefits of a particular model.

4.6 RQ5: Reasons for Not Using Certain UML Diagrams

This section provides answers to research question 5: *What are the reasons for not using function models in software projects?* For this question, only the responses of the 53 UML users were considered, to understand the actual reasons for not using some parts of UML. Respondents were asked to select one or more of the reasons shown in TABLE 16, for each UML diagram they have not used (Q21), hence the different numbers of respondents for each diagram.

Interestingly, the number of respondents providing reasons for not using a diagram is far higher than the number who indicated (see TABLE 16) to have *'never used'* the diagram. These ratios

can be found in Column 1. For example, only two 2 UML users indicated that they never used a sequence diagram, yet 13 UML users provide a reason for not using a sequence diagram. We assume that even though they are users, these respondents still had issues with a particular diagram and expressed this through their answer to this question. Although unintended, the survey thus captured a richer picture of reasons against using a particular diagram. The numbers are low, in particular for the most used diagrams – as can be expected -, but certain patterns emerged.

In TABLE 16, reasons mentioned by more than 40% and 30% of respondents have been highlighted in dark grey and light grey respectively. The models have been categorized in line with the findings described in Section 4.3.1 as Most Used (Dark blue) – Moderately Used (Medium blue) – Less/Never Used (Light blue).

n= Number of Respondents/ Number of '*never used*' Respondents	**Too Complex to Communicate with Clients**	**Information Captured is Redundant**	**Time-consuming**	**Provide no Benefit for Most Projects**	**Lack of Familiarity**	**Other**	**Total Responses**	**Average pp.**
Sequence (n=13/2)	2	5	4	1	2	-	14	1.1
Use Case (n=10/3)	3	2	4	1	1	1	12	1.2
Class (n=14/3)	5	2	7	2	2	1	19	1.4
Activity (n=15/7)	2	4	6	2	4	-	18	1.2
Component (n=20/11)	3	7	4	2	8	-	24	1.2
Deployment (n=19/10)	3	3	3	6	8	-	23	1.2
Object (n=28/15)	8	7	5	5	11	1	37	1.3
State Chart (n=27/11)	4	6	6	3	13	-	32	1.2
IOD (n=31/23)	1	6	5	4	18	1	35	1.1
Use Case Narrative (n=23/19)	-	7	4	4	11	2	28	1.2
Package (n=27/22)	2	9	4	5	11	-	31	1.1
Communication (n=31/21)	3	8	6	3	15	1	36	1.2
Timing (n=30/27)	4	5	8	2	19	1	39	1.3
CSD (n=31/27)	3	4	5	5	17	-	34	1.1

TABLE 16: Reasons for Not Using Certain UML Diagrams in Software Projects (Multiple Reasons Possible) (Dark Grey = More than 40% of Respondents, Light Grey = More than 30% of Respondents, Blue = Categorization Most Used (Dark) – Moderately Used (Medium)– Less/Never Used (Light), see Section 4.3.1)

Too complex to communicate with clients: For three diagrams this reason for not using a diagram was mentioned more often than for other diagrams: use case (30.0% of respondents), class (35.7%), and object (28.6%) although the actual number of respondents is low (3, 5, and 8 respectively). Against our expectations, no respondent considered use case narratives too complex, even though both use cases and use case narratives were widely used for requirement communication with clients (see Section 4.5.1). As the numbers are low for the less used diagrams, complexity does not seem to be a reason for not using those.

Information captured is redundant: The sequence diagram (38.5%), component diagram (35.0%), use case narrative (30.4%), and package diagram (33.3%) were mentioned most often

for containing redundant information. Having this as a reason for sequence diagrams is interesting, as this is one of the most used diagrams (see Section 4.3.1). The number of responses, however, is low (n=5). When considering the response counts, package and communication diagrams show the highest values with 9 and 8 responses, respectively, whereas class and use case diagrams received the lowest with 2 responses each. Comparing the diagrams, some are very similar indeed, only differing in representation, e.g. class and object, sequence and communication, timing, and state chart. Of the 14 diagrams, most system features can be covered using 4-5 diagrams.

Time-consuming: Surprisingly this reason is highlighted for all 4 mostly used diagrams; sequence, use case, class, and activity (31%-50% of their total responses). For all the other diagrams this seems to be an infrequent reason for not using. However, based on the number of respondents, class and timing diagrams were considered time consuming with 8 and 9 responses, respectively.

Provide no benefits for most projects: This is a less frequently mentioned barrier for using UML, as the highest values are stated for deployment diagram with 31.6%, while object, package, and CSD diagrams received 16% -19%.

Lack of familiarity: This appears the most selected reason with 39%-64% of responses, except for the most used diagrams (sequence, use case, class, and activity diagrams) as can be expected. More than 54% of respondents mentioned a lack of familiarity with IOD, timing, and CSD. These are also the diagrams Reggio et al. (Reggio et al., 2013) found least often in educational material.

Other: Only one or two respondents stated that they have other reasons than the predefined ones, for not using use case, object, IOD, communication, timing diagrams and use case narratives.

The results highlight three major, very different barriers to use UML in designing; a lack of familiarity, the time required to use the diagram, closely followed by the opinion that the diagram captures redundant information.

Among these, the lack of familiarity was the main reason (in number of respondents) for 10 diagrams. Education may be a reason or the preference in the industry for the most popular models. Earlier studies (Dobing & Parsons, 2008; Grossman et al., 2005; Wrycza & Marcinkowski, 2007), highlighted complexity, inconsistency, and understandability as the main reasons for not using UML. Our study is more fine-grained in that we found differences between the UML diagrams. The fact that the studies were conducted at different times and involved different professions may also have contributed to the differences in findings.

As concerns the profession, all professions use all diagrams (as shown in FIGURE 3) but we found that analysts considered every diagram time-consuming, whereas respondents from academia considered only sequence, class, activity, and state chart as time-consuming. Software developers considered all 4 most used diagrams as well as component, deployment, and object diagrams as time-consuming. Analysts considered diagrams like class, object, and component as difficult to use when communicating with clients. Importantly, a considerable number of developers mentioned a lack of familiarity with the most widely used behavioral diagrams like use case, activity, state chart diagrams, and use case narratives. This may be because they usually do not participate much in the requirement analysis phase. Finally, apart from software developers, the other respondents mentioned redundancy of information in activity and sequence diagrams as an issue, even though they use these frequently.

4.7 RQ6: The Influence of the Use of SDLC Models on Function Model Usage

In the study, the respondents were asked open-ended questions about the SDLC models they use, personally (survey question Q5) and in the organization (survey question Q6) to answer

research question 6: *What are the currently used SDLC models (personal and organizational)? Do the personal SDLC models have an influence on function model usage?*

For the personal SDLC practices, the 75 respondents together provided a total of 109 responses to this open-ended question, while for other organizational practices only 29 respondents replied, resulting in 34 responses. Only 6 models were mentioned. In both cases, the majority of respondents mentioned that they use agile models in their projects as well as in their organizations, although the percentages differ strongly (see TABLE 17 and TABLE 18).

For personal use, 97.3% (n=73) mentioned agile models, followed by 34.7% (n=26) for the waterfall model. For other organizational SDLC models, 44.8% (n=13) mentioned agile and 41.4% (n=12) the waterfall model. These results can be correlated with the findings in Section 4.3.4, as 23 (of 39 who use non-UML function modelling approaches) mentioned that they use user stories which are introduced with agile practices, which have been increasing strongly. We, therefore, conclude that user stories ranked highest in the non-UML diagrams due to the growth of agile programming and possibly the increased emphasis on user experience and user-centered design. For both personal and other organizational use, very few mentioned Unified process (UP), Spiral, Iterative, and V model.

SDLC Model	# of Respondents	Percentage
Agile	73	97.3%
Waterfall	26	34.7%
Unified Process	4	5.3%
Spiral	4	5.3%
Iterative	1	1.3%
V model	1	1.3%

TABLE 17: Personal Use of SDLC Models (n=75).

SDLC Model	# of Respondents	Percentage
Agile	13	44.8%
Waterfall	12	41.4%
Unified Process	5	17.2%
Spiral	2	6.9%
Iterative	1	3.5%
V model	1	3.5%

TABLE 18: Other SDLC Models Used in Respondents' Organizations (n=29).

FIGURE 9 combines the answers to survey question Q5: "*Which SDLC models have you used or are using currently?"* (see TABLE 17) and survey question Q8: "*Which function modelling approach have you used or are using currently?"* (see TABLE 9). The data suggest that almost every function modelling approach can be combined with different SDLC models. Importantly, even though at the time of its introduction UML was recommended for the 'Unified Process' SDLC model, UML now seems to have been used with every SDLC model. Grossman et al. (Grossman et al., 2005) found similar results. Flow charts, DFD, and block diagrams also seem to be compatible with different SDLC models, even though they were originally intended for use with waterfall models. A fair number of users (n=10, 13%) use SDLC models, but no function modelling approach. It can be concluded that even though most function modelling approaches were introduced with specific SDLC models, with time people have found ways to apply these approaches in almost all the SDLC models available.

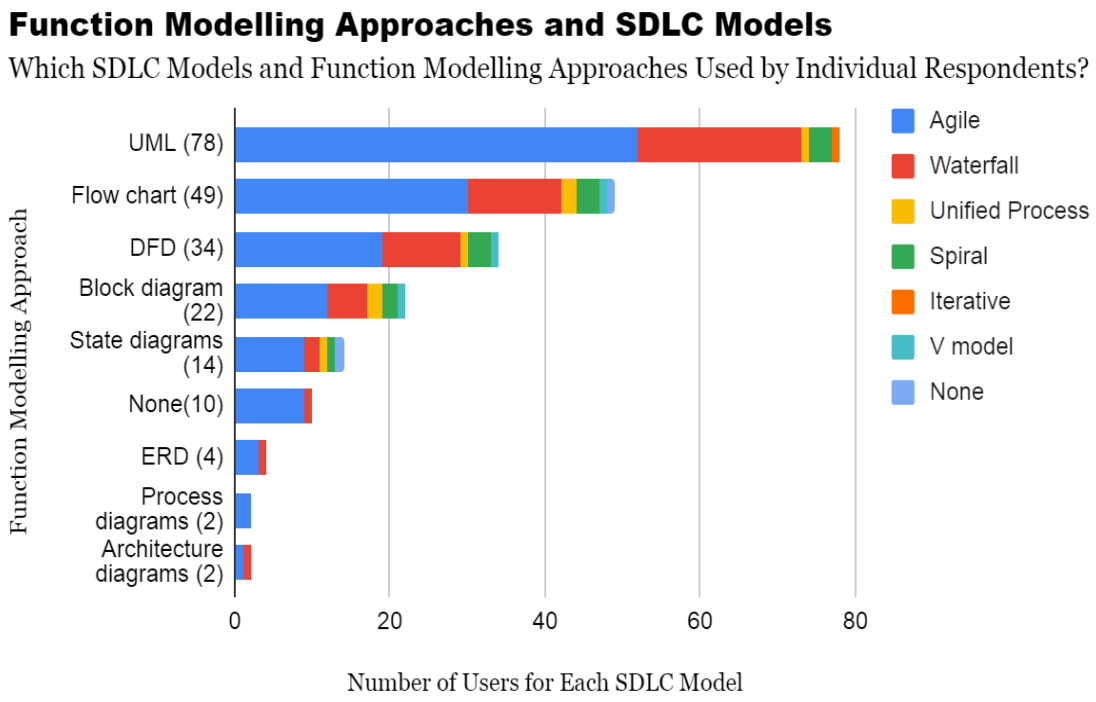


FIGURE 9: Function Modelling Approaches Used in Conjunction with SDLC Models (n=75).

4.8 RQ7 and RQ8: Importance of Function Models in Software Development and Their Contribution to Project Success

Research question 7: *In which phases of software development are function models considered to be most important?* and research question 8: *How does the use of function models affect the success of a project?* are reflected in survey questions Q24 & Q25. All 75 respondents answered both questions.

In Q24, five software development phases were listed: analysis, design, implementation, testing, and maintenance. The respondents were asked to score the importance of function models for each phase as '*very important, 'important', 'slightly important', 'not at all important'' and 'no opinion*'.

Function models were found to be '*very important'* in all the phases of a software project (see FIGURE 10). The function models were considered particularly important in the design and analysis phases (>95% of respondents selected '*very important'* or '*important*'). The answers also indicate that function models are considered important even after deploying the software. The maintenance phase is the only phase for which some respondents (8%, n=6) considered function models '*not at all important'*.

Q25 asked whether respondents agreed with the statement that software projects that use function modelling techniques have a higher chance of success than other projects. More than 90% of respondents agree ('*Strongly Agree'* (44.0%) or '*Agree*' (46.7%)) (see TABLE 19). Only 9.3% selected '*Neutral*'. Interestingly, none of the 75 respondents selected either '*Disagree*' or '*Strongly Disagree*'.

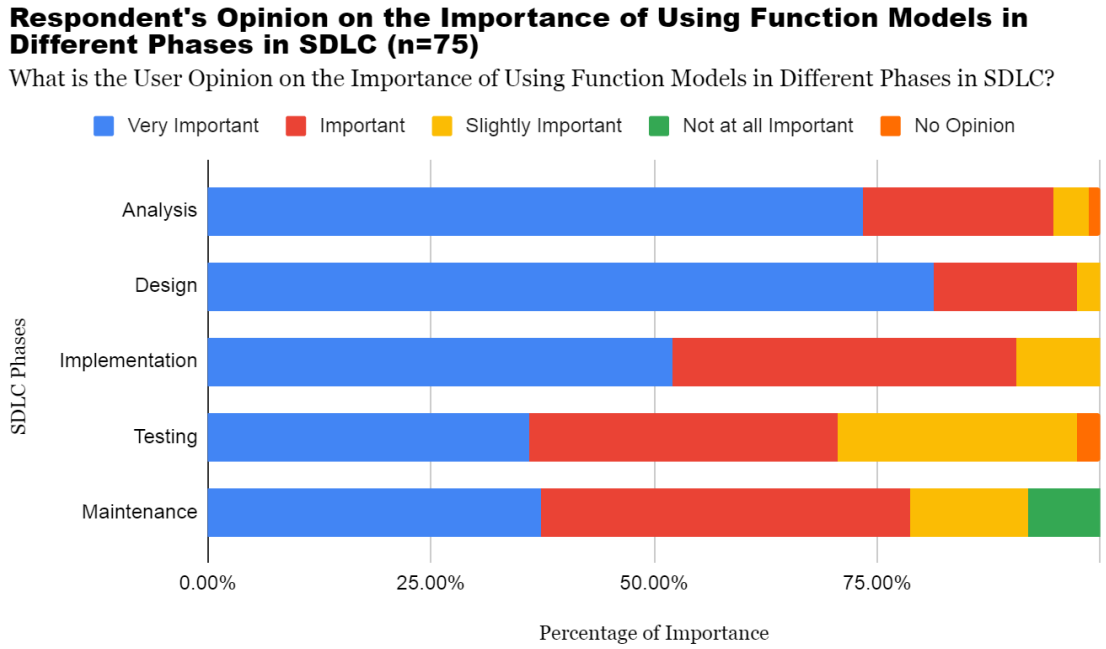


FIGURE 10: Importance of Function Models in Different Software Development Phases (n=75).

Do Software Projects that Use Function Modelling Approaches Have a Higher Chance of Success than Other Projects?	# of Respondents	Percentage
Strongly Agree	33	44.0%
Agree	35	46.7%
Neutral	7	9.3%
Disagree	-	-
Strongly Disagree	-	-
Total	**75**	**100.0%**

TABLE 19: Use of Function Modelling Approaches and its Impact on Project Success (n=75).

5. SUMMARY AND DISCUSSION OF THE RESULTS

5.1 Summary of the Application of Function Models in Practice

The study delivers a deeper understanding of the use of function models within the software community in industry and academia. We were able to answer all eight research questions.

For research question 1, on the function models that are used for software design, we found that various function modelling approaches were used. UML appeared to be the most widely used approach (mentioned by 75% of our respondents). Interestingly, a considerable number of users still use function models introduced before the 1990's such as DFD, flowcharts, block diagrams, and ERD. These results contradict with some previous surveys (Akdur et al., 2018; Farias et al., 2018; Petre, 2013). Over 70% of their respondents mentioned not using UML or any other formal function modelling approach, whereas 93% of our respondents does use function models throughout the SDLC. During conceptual design 70% use only UML and 53% use both UML and other approaches. Despite these high numbers, our data suggests that organizations do not promote UML: only a quarter of our respondents use UML consistently for all their projects.

Research question 2 focused on the application of function models specifically in the conceptual design phase. We analyzed the results for UML and for other approaches separately. For UML, most previous studies (see TABLE 11) found that sequence, use case and class diagrams were the most used diagrams, while IOD, CSD, and timing diagrams were used least. Our study differed in that we considered all 14 UML diagrams and that the majority of our respondents used UML. Nevertheless, our findings are very similar: only 4 of the 14 available diagrams - sequence, use case and class, as well as activity diagrams - are applied frequently. It seems that the application of UML diagrams has not changed much over the past 14 years, even though UML is considered a trend among users today (Ozkaya, 2019). In contrast to previous studies, we not only investigated the application of UML, but also the use of other modelling approaches. We found 7 other function modelling approaches used in the conceptual design stage (see TABLE 13). Among these, user stories stood out compared to approaches such as DFD, ERD, flow charts, etc. Possible reasons are the increase of agile software development, which emphasizes user stories, and the fact that these are considered easy to understand and useful to identify the most urgent features to be implemented (Hvatum & Wirfs-Brock, 2018). Our data suggests that software developers and testers prefer structural models, such as UML class, object, and component diagrams, ERD, and block diagrams, while software analysts tend to use behavioral and interaction diagrams, such as UML use case, sequence diagrams, DFD, and user stories.

Research question 3 aimed at a better understanding of the importance our respondents assign to each UML diagram in conceptual design, irrespective of whether they actually use the diagram. Previously surveys only focused on the use of UML, not on the importance of the individual diagrams. As expected, actual use and importance showed a similar pattern (use case, sequence, class, and activity diagrams were both the most used and most important), the respondents considered more diagrams important than those they used. Our findings are in line with the study of Waykar (Waykar, 2013) who found the same four diagrams as very important in the early design stages.

To answer research question 4 on the purpose of using function models, we separately analyzed UML and other function modelling approaches, as we expected different results. The 3 key purposes for using UML appear to be to verify and validate requirements with clients, specify requirements for programmers, and collaborate between technical teams. The behavioral and interaction diagrams are mainly used for client communication, whereas for internal communication these and structural diagrams are used. UML was found to be used less for activities such as documentation and code generation. Our results are in line with earlier findings of (Dobing & Parsons, 2008; Petre, 2013), suggesting that little has changed, but contradict with (Aranda, 2010) who found function diagrams to be ignored by everyone except software analysts. As to the purpose of non-UML function models, literature seems scarce. In contrast to our expectations, the purposes for non-UML models and UML models is similar: requirement-related purposes and communication. Since this was an open-ended question unlike for UML, we found a few more purposes although with few responses. Again, user stories stood out from the other six models: their main purpose was for communicating requirements with clients and programmers, while the other 6 models had purposes specific to the characteristics of the respective model.

Reasons for using a particular model are not necessary the opposite of reasons for not using a model. The latter was addressed through research question 5. As mentioned before, only four UML diagrams were used frequently. Our data suggests 3 key reasons why certain diagrams are not used: a lack of familiarity, the time required to use the diagram, and redundancy of information captured in the diagram. Lack of familiarity was mentioned for all but the four most used diagrams. A possible explanation is that users tend to learn only the most popular diagrams (some studies aimed at identifying the most used diagrams to inform education) and do not investigate the use of other available diagrams. Our results are slightly different from the previous studies, which highlighted complexity, inconsistency, incompleteness, and lack of understandability as the main reasons for not using UML (Dobing & Parsons, 2008; Grossman et al., 2005; Wrycza & Marcinkowski, 2007). Another mentioned reason is that codes or other

software artifacts generated from UML are not usable (Petre, 2013; Wrycza & Marcinkowski, 2007).

We were also interested in the currently used SDLC models and their influence on the use of function model (Research question 6). Most function modelling approaches were developed and practiced within the context of a particular SDLC model (Capretz, 2003), e.g. UML with the Unified Process. The continuous introduction of agile methods such as Scrum, Kanban, Lean, etc. over the past decade (Al-Zewairi et al., 2017) could thus explain our finding that agile development is by far the most popular approach (mentioned 97% of respondents) compared to traditional models such as Waterfall, Spiral, and V models. Nevertheless, our results also suggest that today function modelling approaches are compatible with different SDLC models. Grossman et al. (Grossman et al., 2005) found similar results for UML applications.

Research question 7 aimed at identifying the importance of function models in particular phases of software development. Our data clearly shows the importance given to function models in the analysis and design phases of software development projects, rather than in the implementation and testing phases. This suggests that industry and academia understand the necessity to clarify system requirements and functionality as early as possible among all involved, instead of starting coding without ensuring this clarity. We did notice, however, a slight difference our findings and those of (Lange et al., 2006) who found implementation, design, and architecting as the phases that require UML the most. Possible reasons are their respondents (software architects) and their focus on UML only.

Finally, in answer to research question 8, we found that the vast majority of our respondents (91%) opine that software projects that use function modelling approaches have a higher chance of success than other projects. This is interesting as it indicates that people believe in the importance of function models in software development, even though our data shows that such models are not consistently used in practice.

5.2 Limitations of the Study

Two limitations might have influenced our study both related to the time available for the study as part of a Master's thesis. Firstly, although large numbers can be reached using a survey, surveys do not allow the flexibility to adapt questions and flow depending on the answers, and answers tend to be shorter, potentially missing details, explanations, examples, or experiences. Additional interviews could have brought additional insights. Secondly, given the different response rate per profession, and the observed dependency of some answers on the profession, could have affected the outcome. Although a different distribution process or a wider distribution could have led to a better balance, this issue was not foreseen. However, we feel confident that the responses and sample size (n=75), in combination with earlier related research on an integrated function modelling approach for mechanical engineering (Eisenbart, Gericke, & Blessing, 2017), provides sufficient insight to develop a prototype function modelling approach aimed at mitigating some of the issues with current approaches.

5.3 Towards an Effective and Simple Function Modelling Approach

Today, UML is considered the standard method for conceptual modelling because hierarchical models are not very compatible with the latest programming languages, which are OO-based (Booch et al., 1998; Jilani et al., 2011). Our respondents all agreed about the importance of function models for the success of a software development project. Nevertheless, our study observed some important issues limiting, or even preventing, the use of UML. Neglecting function models may cause unnecessary iterations during and after development, as functionalities and concepts may not have been correctly identified or communicated at the beginning of a project. We believe there is room for an effective and easy-to-use software function modelling approach.

Such function modelling approach ought to be simple yet contain all the necessary elements to develop solution concepts from the identified requirements and easy to maintain throughout the project. Furthermore, such an approach needs to be understandable for everyone involved,

including technical and business users of the system, and flexible enough to allow different viewpoints based on the user's profession (Kruchten, 1995). The approach should also be able to solve the difficulties faced when using multiple individual models during conceptual design.

Our study highlights that even though numerous function models are known in industry, only a selected group has been prominently practiced. Among these, UML models seem to be the most widespread, but DFD, flowcharts, ERD, and user stories are also popular during conceptual design. We found that behavioral and interactive UML diagrams such as sequence, activity, use case, and state chart are more popular than structural diagrams. Only class diagrams are used frequently when it comes to visualizing the structure of the system.

Interestingly, the key individual features of the widely used function models, are very similar even though they are represented using different notations and belong to different design paradigms. For example, most behavioral function models consist of *activities, functions, data* or *material flows,* and *states,* while the structural models consist of *components* or *entities* with their *links*. In OO design, system entities and components are considered in a common form, the '*object*'. The '*use cases'* that are used to describe the behavior of the system have become a standard and became much simpler in agile practice in the form of a '*user story'*.

The shared features of all the widely used function models are identified as follows:

- UML class diagram and ERD – Represent the <u>structure</u> of the system, including all the data stores, entities/objects, and their relationships.
- DFD, UML activity diagrams, and flow charts – Represent the <u>flow of a set of functions/activities and data</u> which need to be attended to to complete a scenario of the system.
- Use cases and user stories – Represent all the <u>ways that the system can be used by potential users/actors</u> and explain software features from the perspective of the end-users.
- UML sequence diagram and class diagram – Show the <u>interactive message/method calls</u> between objects and actors.
- State chart diagrams – Represent the <u>state changes</u> of each object/entity concerning the activities performed during the process flow.

These common features can be considered as the most important and beneficial features of an approach to model software. Therefore, ***use cases, processes/activities, states, data and material, objects, and users (actors)*** are '*must include'* features of an effective and easy-to-use function modelling approach. They comprise both behavioral and structural aspects of the system while providing the fundamentals for developing a comprehensive architecture of the system.

The function modelling approach we have developed is based on this foundation. It combines the essential features of current function models to encourage the use of multiple function models in early design stages in order to develop more successful software (Abayatilake, 2021). This approach will be published separately.

6. CONCLUSIONS

In this paper, we have presented results from a survey conducted within the software community including both practitioners and people from academia, to investigate the application of function models in the conceptual design stage. We examined the different function modelling approaches that were used, reasons and barriers to their use, their perceived importance and impact on project success. Our findings show that UML is the most widely used and popular function modelling approach among the 75 respondents, compared to structured, data-driven, and agile function models. However, only four of 14 available UML diagrams (sequence, use case, class, and activity diagrams) appear to be frequently used during the conceptual design phase, and with the growth of agile software development practices, UML diagrams seems to now be in competition with agile-specific models. The key reasons to use function models (UML or other)

were found to be to verification and validation of requirements with clients as well as communication and collaboration with software developers and technical teams. The main reasons for not using UML models in the conceptual design stage were first and foremost lack of familiarity, but also time required, and information redundancy. The study shows the selection and application of models greatly depends on the user's profession: in general, non-technical users (analysts) prefer behavioral and interaction function models, while technical users (e.g. developers and testers) tend to use structural models. The study also highlighted that people do believe in the importance of function models for the entire SDLC, especially during the analysis and design phases, and their impact on a successful software project.

This survey allowed us to identify the key features of and issues with the existing function models, and highlighted the opportunity to develop a novel, easy-to-use and efficient software function modelling approach that organizes all the beneficial modelling features into a single framework. These features are: use cases, processes/activities, states, data and material, objects, and users (actors).

Even though several authors have analyzed function models and their application in software development, most of the studies are limited to the models of one design paradigm only, in particular UML. We also included other function modelling approaches and models mentioned by our respondents resulting in a broader picture of the use of and issues with function models and hence providing additional insights into functional modelling and further areas of study. Our study also contributes to practice. The survey showed that model selection always depends on project type and user's task and profession, emphasizing that in most software design projects, multiple function models have to be developed and maintained. This raises issues like maintaining consistency, avoiding duplicate efforts, and ensuring communication. We believe that a novel function modelling approach is necessary to address these types of issues by assembling all the identified essential features of the widely used function models into one framework. This should have the additional benefit of more effective collaboration and a common understanding across all those involved in a project, while visualizing the complete system for all SDLC phases.

Finally, with the advent of industry 4.0, our results provide a necessary complement to the functional modelling efforts in mechanical engineering. Our new approach uses the same structure as our function modelling approach in mechanical engineering (Eisenbart et al., 2017) in order to realize our aim of a function modelling approach that can represent both hardware and software elements of a system.

Based on our survey, we conclude that such approach requires the following:

- All the identified key features need to be included to form a comprehensive function modelling approach. This requires a new type of representation to enable the inclusion of all features in one diagram.
- To be compatible with the current growth of agile programming, function models need to be much simpler, faster, and easily understandable, in order to allow for a rapid prototyping and feedback cycle.
- To explain the external user interactions with the system, OO concepts rather than traditional structured and DD (data-driven) function modelling concepts, are preferred, as the traditional concepts only focus on internal processes and data flows.
- Structural and behavioral function models are equally important in transforming requirements into a conceptual design, as each model represents different aspects of the system, and all aspects are important in finding successful and alternative or exceptional solution paths.
- Process flows must be included because they are essential for conceptual design to describe the fundamental activities of the system and to show interactions between different system entities.

Combining the identified modelling features into one, easy-to-use function modelling approach is – to our knowledge – novel to software design. We believe that such an approach will motivate

designers and developers to more comprehensively use functional modelling in the conceptual design stage and increase project success.

7. ACKNOWLEDGEMENTS

This research was enabled by a Master of Engineering in Innovation by Design (MIBD) scholarship of Singapore University of Technology and Design.

8. REFERENCES

Abayatilake, P. (2021). Function Modelling: Benefits and Limitations of Current Models for Complex Software Development. (Master's thesis). Singapore University of Technology and Design,

Agarwal, R., De, P., & Sinha, A. P. (1999). Comprehending object and process models: an empirical study. IEEE Transactions on Software Engineering, 25(4), 541-556. doi:10.1109/32.799953

Akdur, D., Garousi, V., & Demirörs, O. (2018). A survey on modeling and model-driven engineering practices in the embedded software industry. Journal of Systems Architecture 91, 62-82. doi:10.1016/j.sysarc.2018.09.007

Al-Zewairi, M., Biltawi, M., Etaiwi, W., & Shaout, A. (2017). Agile Software Development Methodologies: Survey of Surveys. Journal of Computer and Communications, 05(05), 74-97. doi:10.4236/jcc.2017.55007

Ambler, S. W. (2004). The Object Primer: Agile Model-Driven Development with UML 2.0. 40 W. 20 St. New York, NY, United States: Cambridge University Press.

Anwar, A. (2014). A Review of RUP (Rational Unified Process). International Journal of Software Engineering (IJSE), 5(2), 8-24.

Aranda, J. (2010). A theory of shared understanding for software organizations. (PhD thesis). University of Toronto,

Atan, R., Ghani, A. A. A., Selamat, M. H., & Mahmod, R. (2007). Automating Measurement for Software Process Models using Attribute Grammar Rules. International Journal of Engineering (IJE), 7(2), 24-33.

Baldwin, A. N., Austin, S. A., Hassan, T. M., & Thorpe, A. (1999). Modelling information flow during the conceptual and schematic stages of building design. Construction Management and Economics, 17(2), 155-167. doi:10.1080/014461999371655

Booch, G. (1994). Object-Oriented Analysis and Design With Applications (2 ed.). Redwood City, CA, United States: Addison Wesley.

Booch, G., Rumbaugh, J., & Jacobson, I. (1998). The Unified Modeling Language User Guide. Boston, MA, United States: Addison Wesley.

Capretz, L. F. (2003). A brief history of the object-oriented approach. ACM SIGSOFT Software Engineering Notes, 28(2). doi:10.1145/638750.638778

Champeaux, D. d., Constantine, L., Jacabson, I., Mellor, S., Ward, P., & Yourdon, E. (1990). PANEL: Structured Analysis and Object Oriented Analysis. Paper presented at the OOPSLA/ECOOP '90: Proceedings of the European conference on Object-oriented programming addendum : systems, languages, and applications: systems, languages, and applications.

Chren, S., Buhnova, B., Macak, M., Daubner, L., & Rossi, B. (2019). Mistakes in UML Diagrams: Analysis of Student Projects in a Software Engineering Course. Paper presented at the 2019

IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), Montreal, QC, Canada.

Colquhoun, G. J., Baines, R. W., & Crossley, R. (1993). A state of the art review of IDEFO. International Journal of Computer Integrated Manufacturing, 6(4), 252-264. doi:10.1080/09511929308944576

Cooper, A., Reimann, R., & Cronin, D. (2007). About face 3: the essentials of interaction design. 605 Third Ave. New York, NY, United States: John Wiley & Sons, Inc.

Dalpiaz, F., & Brinkkemper, S. (2018). Agile Requirements Engineering with User Stories. Paper presented at the 2018 IEEE 26th International Requirements Engineering Conference (RE), Banff, AB, Canada.

Dobing, B., & Parsons, J. (2008). Dimensions of UML Diagram Use. Journal of Database Management, 19(1), 1-18. doi:10.4018/jdm.2008010101

Dorador, J. M., & Young, R. I. M. (2000). Application of IDEF0, IDEF3 and UML methodologies in the creation of information models. International Journal of Computer Integrated Manufacturing, 13(5), 430-445. doi:10.1080/09511920050117928

Eisenbart, B., Gericke, K., & Blessing, L. T. M. (2017). Taking a look at the utilisation of function models in interdisciplinary design: insights from ten engineering companies. Research in Engineering Design, 28(3), 299-331. doi:10.1007/s00163-016-0242-3

El-Attar, M., & Miller, J. (2006). Producing robust use case diagrams via reverse engineering of use case descriptions. Software & Systems Modeling, 7(1), 67-83. doi:10.1007/s10270-006-0039-3

Erden, M. S., Komoto, H., van Beek, T. J., D'Amelio, V., Echavarria, E., & Tomiyama, T. (2008). A review of function modeling: Approaches and applications. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 22(2), 147-169. doi:10.1017/s0890060408000103

Erickson, J., & Siau, K. (2007). Can UML Be Simplified? Practitioner Use of UML in Separate Domains. Paper presented at the Proceeding EMMSAD 2007, Trondheim, Norway.

Falessi, D., Cantone, G., & Grande, C. (2007). A Comparison of Structured Analysis and Object Oriented Analysis - An Experimental Study. Paper presented at the Proceedings of the Second International Conference on Software and Data Technologies, Barcelona, Spain.

Farias, K., Gonçales, L., & Bischoff, V. (2018). On the UML Use in Brazilian Industry: A State of the Practice Survey. Paper presented at the SEKE - 30th International Conference on Software Engineering & Knowledge EngineeringAt: San Francisco, CA, San Francisco, CA.

Firesmith, D. (1991). Structured Analysis and Object-Oriented Development are not Compatible. ACM SIGAda Ada Letters, XI(9), 56-66. doi:10.1145/122012.122013

Gregersen, H., & Jensen, C. S. (1999). Temporal Entity-Relationship Models—a Survey. IEEE Transactions on Knowledge and Data Engineering, 11(3), 464-497. doi:10.1109/69.774104

Grossman, M., Aronson, J. E., & McCarthy, R. V. (2005). Does UML make the grade? Insights from the software development community. Information and Software Technology, 47(6), 383-397. doi:10.1016/j.infsof.2004.09.005

Hvatum, L. B., & Wirfs-Brock, R. (2018). A Program Backlog Story with Patterns: Expanding the Magic Backlog Pattern Collection. Paper presented at the EuroPLoP '18: Proceedings of the 23rd European Conference on Pattern Languages of Programs.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). The Unified Software Development Process. Boston, MA, United States: Addison-Wesley.

Jilani, A. A. A., Usman, M., & Nadeem, A. (2011). Comparative Study on DFD to UML Diagrams Transformations. World of Computer Science and Information Technology Journal(WCSIT), 1(1), 10-16.

Kaur, S. (2017). COMPARATIVE ANALYSIS OF SOFTWARE DEVELOPMENT MODELS. INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY, 6(10), 611-617. doi:10.5281/zenodo.1036644

Khan, M. E., Shadab, S. G. M., & Khan, F. (2020). Empirical Study of Software Development Life Cycle and its Various Models. International Journal of Software Engineering (IJSE), 8(2), 16-26.

Kobryn, C. (2002). Will UML 2.0 Be Agile or Awkward? Communications of the ACM, 45(1), 107-110. doi:https://doi.org/10.1145/502269.502306

Kruchten, P. (1995). Architectural Blueprints—The “4+1” View Model of Software Architecture. IEEE Software, 42-50.

Lange, C. F. J., Chaudron, M. R. V., & Muskens, J. (2006). In Practice: UML Software Architecture and Design Description. IEEE Software, 23(2), 40-46. doi:10.1109/ms.2006.50

Modi, H. S., Singh, N. K., & Chauhan, H. P. (2017). Comprehensive Analysis of Software Development Life Cycle Models. International Research Journal of Engineering and Technology (IRJET), 4(6).

Ozkaya, M. (2019). Are the UML Modeling Tools Powerful Enough for Practitioners? A Literature Review. IET Software, 13(5), 338-354. doi:10.1049/iet-sen.2018.5409

Ozkaya, M., & Erata, F. (2020). A Survey on the Practical Use of UML for Different Software Architecture Viewpoints. Information and Software Technology, 121(4). doi:10.1016/j.infsof.2020.106275

Petre, M. (2013). UML in practice. Paper presented at the ICSE '13: Proceedings of the 2013 International Conference on Software Engineering, San Francisco, CA, USA.

Reggio, G., Leotta, M., & Ricca, F. (2014). Who Knows/Uses What of the UML: A Personal Opinion Survey. In Model-Driven Engineering Languages and Systems (pp. 149-165).

Reggio, G., Leotta, M., Ricca, F., & Clerissi, D. (2013). What are the used UML diagrams? A Preliminary Survey. Paper presented at the Proceedings of 3rd International Workshop on Experiences and Empirical Studies in Software Modeling (EESSMod 2013 co-located with MoDELS 2013).

Rickman, D. M. (2001). A process for combining object oriented and structured analysis and design. Paper presented at the 20th Digital Avionics Systems Conference

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). The Unified Modelling Language Reference Manual. Boston, MA. United States: Addison Wesley.

Rumpe, B. (2017). Agile Modeling with UML (1 ed.): Springer International Publishing.

Strauss, A., & Corbin, J. M. (1997). Grounded theory in practice: Sage Publications.

Waykar, Y. (2013). A Study of Importance of UML diagrams: With Special Reference to Very Large-sized Projects. Paper presented at the International Conference on Reinventing Thinking beyond boundaries to Excel, Faridabad, India.

Wieringa, R. (1998). A Survey of Structured and Object-Oriented Software Specification Methods and Techniques. ACM Computing Surveys (CSUR), 30(4), 459-527. doi:10.1145/299917.299919

Wrycza, S., & Marcinkowski, B. (2007). A Light Version of UML 2: Survey And Outcomes. Paper presented at the Computer Science and IT Education Conference, Port Louis, Mauritius.

INSTRUCTIONS TO CONTRIBUTORS

The International Journal of Software Engineering (IJSE) provides a forum for software engineering research that publishes empirical results relevant to both researchers and practitioners. IJSE encourage researchers, practitioners, and developers to submit research papers reporting original research results, technology trend surveys reviewing an area of research in software engineering and knowledge engineering, survey articles surveying a broad area in software engineering and knowledge engineering, tool reviews and book reviews. The general topics covered by IJSE usually involve the study on collection and analysis of data and experience that can be used to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies. IJSE is a refereed journal that promotes the publication of industry-relevant research, to address the significant gap between research and practice.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Started with Volume 9, 2021, IJSE appears with more focused issues. Besides normal publications, IJSE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

We are open to contributions, proposals for any topic as well as for editors and reviewers. We understand that it is through the effort of volunteers that CSC Journals continues to grow and flourish.

IJSE LIST OF TOPICS

The realm of International Journal of Software Engineering (IJSE) extends, but not limited, to the following:

- Ambiguity in Software Development
- Architecting an OO System for Size Clarity Reuse E
- Computer-Based Engineering Techniques
- History of Software Engineering
- Impact of CASE on Software Development Life Cycle
- Iterative Model
- Licensing
- Object-Oriented Systems
- Quality Management
- SDLC
- Software Deployment
-
-
- Software Engineering Demographics
- Software Engineering Methods and Practices
- Software Ergonomics
- Structured Analysis
- Systems Engineering
- UML
- Application of Object-Oriented Technology to Engin
- Composition and Extension
- Data Modeling Techniques
- IDEF
- Intellectual Property
- Knowledge Engineering Methods and Practices
- Modeling Languages
- Project Management
- Rational Unified Processing
- Software Components
- Software Design and applications in Various Domain
- Software Engineering Economics
- Software Engineering Professionalism
- Software Maintenance and Evaluation
- Structuring (Large) OO Systems
- Test Driven Development
-

CALL FOR PAPERS

Volume: 9 - **Issue: 3**

i. Submission Deadline : October 31, 2021 **ii. Author Notification:** November 30, 2021

iii. Issue Publication: December 2021

CONTACT INFORMATION

Computer Science Journals Sdn BhD

B-5-8 Plaza Mont Kiara, Mont Kiara

50480, Kuala Lumpur, MALAYSIA

Phone: 006 03 6204 5627
Fax: 006 03 6204 5628
Email: cscpress@cscjournals.org

CSC PUBLISHERS © 2021
COMPUTER SCIENCE JOURNALS SDN BHD
B-5-8 PLAZA MONT KIARA
MONT KIARA
50480, KUALA LUMPUR
MALAYSIA

PHONE: 006 03 6204 5627
FAX: 006 03 6204 5628
EMAIL: cscpress@cscjournals.org