# Augmenting software engineers with modeling assistants

Maxime Savary-Leblanc

Université de Lille


**University of Lille**

Doctoral School **MADIS**

Laboratory **CRIStAL UMR 9189**


Thesis defended by **Maxime Savary-Leblanc**

Defended on **15th December, 2021**

In order to become Doctor from University of Lille

Academic Field **Computer Science**

# Augmenting software engineers with modeling assistants

**Thesis supervised by**

| | |
|---|---|
| Sébastien Gérard | Co-Supervisor |
| Xavier Le Pallec | Co-Supervisor |

**Committee members**

| | | |
|---|---|---|
| *Referees* | Jeff Gray | Professor at University of Alabama |
| | Dimitris Kolovos | Professor at University of York |
| | Ileana Ober | Professor at Université Paul Sabatier |
| *Examiners* | Silvia Abrahao | Associate Professor at Universitat Politècnica de València |
| | Célia Martinie | HDR Associate Professor at Université Paul Sabatier |
| | Lionel Seinturier | Professor at Université de Lille<br>Committee president |
| *Supervisors* | Sébastien Gérard | Senior Researcher at CEA List |
| | Xavier Le Pallec | HDR Associate Professor at Université de Lille |

Université de Lille


**University of Lille**

Doctoral School **MADIS**

Laboratory **CRIStAL UMR 9189**


Thesis defended by **Maxime Savary-Leblanc**

Defended on **15th December, 2021**

In order to become Doctor from University of Lille

Academic Field **Computer Science**

# Augmenting software engineers with modeling assistants

**Thesis supervised by**

| | |
|---|---|
| Sébastien Gérard | Co-Supervisor |
| Xavier Le Pallec | Co-Supervisor |

**Committee members**

| | | |
|---|---|---|
| *Referees* | Jeff Gray | Professor at University of Alabama |
| | Dimitris Kolovos | Professor at University of York |
| | Ileana Ober | Professor at Université Paul Sabatier |
| *Examiners* | Silvia Abrahao | Associate Professor at Universitat Politècnica de València |
| | Célia Martinie | HDR Associate Professor at Université Paul Sabatier |
| | Lionel Seinturier | Professor at Université de Lille<br>Committee president |
| *Supervisors* | Sébastien Gérard | Senior Researcher at CEA List |
| | Xavier Le Pallec | HDR Associate Professor at Université de Lille |

Université de Lille


**Université de Lille**

École doctorale **MADIS**

Unité de recherche **CRIStAL UMR 9189**

Thèse présentée par **Maxime Savary-Leblanc**


Soutenue le **15 décembre 2021**

En vue de l'obtention du grade de docteur de l'Université de Lille

Discipline **Informatique**

# Supporter les ingénieurs logiciels avec des assistants de modélisation


**Thèse dirigée par** Sébastien Gérard co-directeur
Xavier Le Pallec co-directeur


**Composition du jury**

| | | |
|---|---|---|
| *Rapporteurs* | Jeff Gray | professeur à l'University of Alabama |
| | Dimitris Kolovos | professeur à l'University of York |
| | Ileana Ober | professeur à l'Université Paul Sabatier |
| *Examinateurs* | Silvia Abrahao | mcf à l'Universitat Politècnica de València |
| | Célia Martinie | mcf hdr à l'Université Paul Sabatier |
| | Lionel Seinturier | professeur à l'Université de Lille<br>Président du jury |
| *Directeurs de thèse* | Sébastien Gérard | directeur de recherche au CEA List |
| | Xavier Le Pallec | mcf hdr à l'Université de Lille |

This thesis has been prepared at

**CRIStAL UMR 9189**
Université de Lille
Campus scientifique
Bâtiment ESPRIT
Avenue Henri Poincaré
59655 Villeneuve d'Ascq
France

Web Site `https://www.cristal.univ-lille.fr/`

**Augmenting software engineers with modeling assistants**

**Abstract**

Domain knowledge is a prerequisite to produce software design and implementation tailored to stakeholders' requirements. One common way to formalize that knowledge is achieved through conceptual models, which are commonly used to describe or simulate a system. Acquiring such expertise requires to discuss with knowledgeable stakeholders and/or to get an access to useful documents, which both might not always be easily accessible. In the same time, more model samples can be gathered from multiple sources, what represents an increasing number of already formalized and accessible knowledge pieces. For example, some companies keep archives of internal model repositories. There also exist numerous open source projects that contain models while some modeling tools even offer the possibility to create public projects that are free to browse. Such data sources could be exploited to create domain knowledge that could be provided to software engineers while modeling. To be useful, this knowledge must be of high quality, but must also be well integrated into the software modeling process. The focus of this thesis is to provide a framework to exploit knowledge to assist users of computer-based modeling tools with software modeling assistants. This thesis first introduces our research questions based on a systematic mapping study about software assistants for software engineering, and then focuses on software assistants for modeling. It reports on the design of modeling assistants based on a user-centered approach. We present the conclusions of interviews conducted with experts in modeling, a stage in which requirements are collected. Then, we develop the creation of a prototype modeling knowledge base allowing (i) to create general and specific artificial modeling knowledge, and (ii) to make them available to any software client via recommendations. After introducing the results of an experiment regarding the accuracy of the system, we discuss these preliminary results. Finally, this thesis presents a software modeling assistant implementation integrated to the Papyrus tool, which aims to cognify the UML modeling environment by integrating the previously created knowledge. Our work helps to clarify the need for assistance during software modeling work, presents an initial approach to the design of software assistants for software modeling, and identify research challenges in modeling assistance.

**Keywords:** software assistants, modeling, software engineering, trust, creativity, recommender systems, automation

**CRIStAL UMR 9189**
Université de Lille – Campus scientifique – Bâtiment ESPRIT – Avenue Henri Poincaré – 59655 Villeneuve d'Ascq – France

### Supporter les ingénieurs logiciels avec des assistants de modélisation

**Résumé**

La connaissance du domaine est une condition préalable à la conception et à la mise en œuvre de logiciels adaptés aux exigences des parties prenantes. Une façon courante de formaliser cette connaissance est réalisée par des modèles conceptuels, qui sont couramment utilisés pour décrire ou simuler un système. L'acquisition d'une telle expertise nécessite de discuter avec des parties prenantes bien informées et/ou d'avoir accès à des documents utiles, qui ne sont pas toujours facilement accessibles. Dans le même temps, de plus en plus d'échantillons de modèles peuvent être rassemblés à partir de sources multiples, ce qui représente un nombre croissant d'éléments de connaissance déjà formalisés et accessibles. Par exemple, certaines entreprises conservent des archives de référentiels de modèles internes. Il existe également de nombreux projets open source qui contiennent des modèles, tandis que certains outils de modélisation offrent même la possibilité de créer des projets publics que l'on peut parcourir librement. Ces sources de données pourraient être exploitées pour créer une connaissance du domaine qui pourrait être fournie aux ingénieurs logiciels lors de la modélisation. Pour être utile, cette connaissance doit être de haute qualité, mais doit aussi être bien intégrée dans le processus de modélisation du logiciel. L'objectif de cette thèse est de fournir un cadre pour exploiter les connaissances afin d'aider les utilisateurs d'outils de modélisation informatique avec des assistants de modélisation logicielle. Cette thèse présente d'abord nos questions de recherche basées sur une étude de cartographie systématique sur les assistants logiciels pour l'ingénierie logicielle, et se concentre ensuite sur les assistants logiciels pour la modélisation. Elle rend compte de la conception d'assistants de modélisation basée sur une approche centrée sur l'utilisateur. Nous présentons les conclusions des entretiens menés avec des experts en modélisation, une étape au cours de laquelle les exigences sont recueillies. Ensuite, nous développons la création d'un prototype de base de connaissances en modélisation permettant (i) de créer des connaissances artificielles générales et spécifiques en modélisation, et (ii) de les mettre à disposition de tout client logiciel via des recommandations. Après avoir présenté les résultats d'une expérience concernant la précision du système, nous discutons ces résultats préliminaires. Enfin, cette thèse présente l'implémentation d'un assistant de modélisation logiciel intégré à l'outil Papyrus, qui vise à cognifier l'environnement de modélisation UML en intégrant les connaissances précédemment créées. Notre travail permet de clarifier le besoin d'assistance pendant les travaux de modélisation de logiciels, de présenter une première approche de la conception d'assistants logiciels pour la modélisation de logiciels, et d'identifier les défis de recherche dans l'assistance à la modélisation.

**Mots clés :** assistants logiciels, modélisation, ingénieurie logicielle, confiance, créativité, systèmes de recommandation, automatisation

**CRIStAL UMR 9189**
Université de Lille – Campus scientifique – Bâtiment ESPRIT – Avenue Henri Poincaré – 59655 Villeneuve d'Ascq – France

# Acknowledgements

This manuscript is the result of three years rich in emotions, twists and turns, work, efforts, but also pleasure. First of all, I would like to thank in a general way all the people who accompany me every day, from various horizons, in particular from the associative field, which allow me to live so many different adventures.

First of all, I would like to thank the referees of my thesis, professors Ileana Ober, Jeff Gray, and Dimitris Kolovos, who accepted to read the manuscript and produce detailed and enriching reports. Your precise feedback and remarks allowed me to prepare the defense as well as possible, and to continue improving this manuscript. I would also like to thank Silvia Abrahao, Celia Martinie, and Lionel Seinturier, the examiners of my defense, who accepted to travel to offer their comments and remarks on my work.

I will not find strong enough words to thank Xavier Le Pallec and Sébastien Gérard for the 3 years spent at their side, as my thesis supervisors. I am aware of the exceptional working environment that you were able to provide me, materially, but especially humanly. I regularly bothered you to get answers, opinions, material, and you always reacted quickly and positively. It is necessary to note the incredible series of problems that I had to face since the beginning of the thesis, and until the last days before the defense. When you are unlucky ... you are unlucky. However, I will quote a great wise man "only those who do things have to solve problems". Besides, great wise man, with this manuscript, I emancipate myself as a slave!

I also want to thank the Carbon team at the University of Lille for their technical support and our discussions in meetings. I thank the CEA and the Papyrus development team, who did their best to help me solve the problems I encountered with Papyrus. I would also like to thank the researchers I was able to collaborate with during my thesis. Thanks to Antonio Bucchiarone for the PapyGame adventure,

thanks to Loli Burgueño for our discussions and your always meaningful advices, thanks to Rodi Jolak for allowing me to do this interesting experimentation (one month after starting my thesis), thanks to Jordi Cabot for your vision and your always clear and pragmatic feedbacks, thanks to Philippe Palanque for sharing your vision of HCI with me. Thanks also to Michel Chaudron and Bran Selic for our discussions and your wise feedbacks.

Finally, I would like to particularly thank my close friends and family, who have been my support throughout this winding thesis adventure. First, thanks to Florian and Corentin, my lifelong friends, for their presence in my life. Florian, your daily support, our moments of life, your good mood, your adventures, and simply you, allowed me to find (or sometimes to find again) the strength to go on with this 3 years work. If a thesis mixed with two lockdowns did not succeed in making us crack, I believe that you will unfortunately have to support me for a long time. Corentin, thank you for having decided that a thesis was not enough, and that you had to add the organization of your wedding, and very soon the birth of my godchild to the deal. Our online games, our discussions and our laughs were a strength in which I could draw an incredible positive energy. Thank you to Amélie for supporting me at the beginning of my thesis, in a moment of uncertainty, when you don't know what you should do (or even who you are). Finally, I would like to thank my parents for their unfailing support during all these years. I never really knew how to choose what I wanted to do, so I chose to do everything at the same time, and you were always there. Thank you. Thank you to Delphine and Guillaume, my sister and my brother, for their presence and their daily support.

Ce manuscrit est le résultat de trois années riches en émotions, en rebondissements, en travail, en efforts, mais également en plaisir. Je tiens tout d'abord à remercier de manière générale tous les gens qui m'accompagnent au quotidien, d'horizons variés, notamment du milieu associatif, qui me permettent de vivre autant d'aventures différentes.

En premier lieu, je tiens à remercier rapporteurs de ma thèse, les professeurs Ileana Ober, Jeff Gray, et Dimitris Kolovos, qui ont accepté de lire le manuscrit et de produire des rapports détaillés et enrichissants. Votre feedback précis et vos remarques m'ont permis de préparer au mieux la soutenance, et de continuer à améliorer ce manuscrit. Je tiens également à remercier Silvia Abrahao, Célia Martinie et Lionel Seinturier, les examinateurs de la soutenance, qui ont accepté de faire le

déplacement pour venir proposer leurs commentaires et leurs remarques sur mon travail.

Je ne trouverai pas de mots assez forts pour remercier Xavier Le Pallec et Sébastien Gérard pour les 3 années passées à leurs côtés, comme mes directeurs de thèse. J'ai conscience de l'exceptionnel environnement de travail que vous avez pu me fournir, matériellement, mais surtout au niveau de votre disponibilité. Je vous ai régulièrement embêté pour avoir des réponses, des avis, des opinions, du matériel, et vous avez toujours réagi rapidement et positivement. Il faudra quand même noter l'incroyable série de problèmes que j'ai dû affronter depuis le début de la thèse, et jusqu'aux derniers jours avant la soutenance. Quand on n'a pas de chance ... on n'a pas de chance. Cependant, je citerai un grand sage "il n'y a que ceux qui font des choses qui doivent résoudre des problèmes". D'ailleurs, grand sage, avec ce manuscrit, je m'émancipe en tant qu'esclave !

Je souhaite remercier également l'équipe Carbon à l'université de Lille pour le soutien technique et les discussions en réunions. Je remercie le CEA et l'équipe de développement de Papyrus, qui ont fait leur possible pour m'aider à résoudre les problèmes que je rencontrais avec Papyrus. Je souhaite également remercier les chercheurs avec qui j'ai pu collaborer au cours de ma thèse. Merci à Antonio Bucchiarone pour l'aventure PapyGame, merci à Loli Burgueño pour nos discussions et tes conseils toujours justes de sens, merci à Rodi Jolak pour m'avoir permis de faire cette expérimentation interessante (un mois après avoir commencé ma thèse), merci à Jordi Cabot pour ta vision et tes retours toujours clairs et pragmatiques, merci à Philippe Palanque pour partager ta vision de l'IHM avec moi. Merci également à Michel Chaudron et à Bran Selic pour nos discussions et vos retours avisés.

Enfin, je souhaite particulièrement remercier mes proches, qui ont été mon soutien tout au long de cette sinueuse aventure de thèse. D'abord, merci à Florian et Corentin, mes amis de toujours, pour leur présence dans ma vie. Florian, ton soutien quotidien, nos moments de vie, ta bonne humeur, tes aventures, et tout simplement toi, m'ont permis de trouver (ou de retrouver parfois) la force d'avancer sur ce travail de 3 ans. Si une thèse mixée à deux confinements n'ont pas réussi à nous faire craquer, je crois que tu vas malheureusement devoir me supporter encore un bon moment. Corentin, merci d'avoir jugé qu'une thèse n'était pas suffisante, et qu'il fallait ajouter l'organisation de ton mariage, et très bientôt la naissance de mon (ou ma) filleul(e) à l'affaire. Nos parties en lignes, nos discussions et nos fou-rires ont été une force dans laquelle j'ai pu puiser une énergie positive incroyable. Merci à Amélie pour m'avoir

épaulé au début de ma thèse, dans un moment de flou, où l'on ne sait pas bien ce que l'on doit faire (ni même qui l'on est). Merci enfin à mes parents pour leur soutien sans faille depuis toutes ces années. Je n'ai jamais trop su choisir ce que je voulais, alors j'ai choisi de tout faire en même temps, et vous avez toujours répondu présents. Merci. Merci à Delphine et à Guillaume, ma soeur et mon frère, pour leur présence et leur soutien au quotidien.

# Acronyms

**API** Application Programming Interface. 31, 32, 37, 38, 41, 55–57, 149, 164, 166, 168, 181, 183

**CASE** Computer-Aided Software Engineering. 14

**CI/CD** Continuous Integration/Continuous Delivery. 180

**CSCW** Computer-Supported Cooperative Work. 9, 14

**CSE** Collaborative Software Engineering. 14

**DIKW** Data, Information, Knowledge, and Wisdom. 16

**HCI** Human-Computer Interactions. xxiii, 9, 39, 41, 42, 49, 50, 53, 100, 202, 204

**IDE** Integrated Development Environment. 14, 16, 18, 39, 48, 49, 51, 53, 55, 57

**IPSE** Integrated Programming Support Environments. 14

**MBSE** Model-Based Software Engineering. 1, 4, 22, 85, 129

**MCRS** Multi-Criteria Recommender System. 128, 129, 170

**MDE** Model-Driven Engineering. 28, 68, 69, 85

**ML** Machine Learning. 28–30, 53, 192

**OMG** Object Management Group. 22

**RS** Recommender System. 35, 36, 40, 41, 44, 55, 56

**RSSE** Recommender Systems for Software Engineering. 28

**SASM** Software Assistant for Software Modeling. xix, 122, 139, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 235

**SEE** Software Engineering Environments. 14

**SMS** Systematic Mapping Study. 27, 29

**SWEBOK** Software Engineering Body of Knowledge. 29, 30

**UI** User Interface. 36, 148

**UML** Unified Modeling Language. xxiii, 9, 18–20, 36, 44, 55, 56, 62, 64, 66–74, 85–87, 95, 129, 130, 132, 147–149, 161–163, 166–172, 176, 180, 181, 191, 194, 200, 204

# Summary

# List of Tables

# List of Figures

Chapter 1

# Introduction

## 1.1 Global context

Model-Based Software Engineering (MBSE) is now a widely recognized methodology and its potential benefits are no longer to prove [76]. However, the increasing complexity of software systems to design, and the broad diversity of user profiles to deal with push current modeling tools to their limits [88]. In different studies, the usability or the interoperability of such tools are very often stressed as major factors that slow down the adoption of the model-based approaches. These elements of accidental complexity [9] then increase the required mental effort to conceive models.

In the meantime, current modeling tools still poorly support the essential complexity of the modeling task. This refers to the inherent difficulty of the problem such as knowing and understanding the domain concepts, choosing what to include in the model or not, or selecting the right level of abstraction for a model [186]. While this essential complexity cannot be reduced by definition, it can be addressed to assist users during the creative problem-solving task of modeling. However, the quality or even the existence of such assistance is tightly coupled to the availability and the quality of domain and modeling knowledge.

Additional model samples can be gathered from multiple sources, which represents an increasing number of already formalized and accessible knowledge pieces. For example, some companies maintain internal model repositories [194]. There also exist numerous open source projects that contain models [72] while some modeling tools even offer

the possibility to create public projects that are free to browse. Modeling tools could benefit from this available data to embed new knowledge-empowered features –software modeling assistants– that augment users and help them cope with the essential complexity of modeling. This approach has been investigated in a few different papers [52, 162] and industrial tools[1][2], but is still not mainstream in modeling tools.

## 1.2 Thesis directions

In this thesis, we aim at understanding the modeling task better to propose a solution to support it. Specifically, **we investigate how the notion of *software assistants* could be a solution to tackle current software modeling issues**. In this (less formal) section, we present our vision of the work reported in this thesis manuscript. More specifically, we present how we aim to address the topics of (i) software assistants, (ii) current issues in software modeling, and (iii) solutions to these issues.

### 1.2.1 Software assistants

Software assistants are increasingly present in our lives, whether they are materialized by hardware components, or only virtual, like those integrated in our phones or cars. In fact, the term *software assistant* is regularly used in the media, and consequently in the scientific literature, following the evolution of the common vocabulary. However, to the extent of our knowledge, this term has never been defined in the literature for software modeling nor for software engineering. At the extreme, it seems that this term does not have a consensual definition for everyday-life assistants either. Thus, the use of the term *software assistant* seems to be at the discretion of the authors, judging on their own, in pure subjectivity, if their system qualifies or not for this appellation.

One of the goals of this thesis is to clarify the notion of software assistant for software engineering and software modeling. In this manuscript, we propose our answer to "What is a software assistant for software engineering?", based on the literature. We then show that, although software *modeling* assistants share the same definition as software engineering assistants, they must be designed in a particular way, in order to address the specific problems of the modeling task.

[1] https://www.outsystems.com/ai/
[2] https://www.mendix.com/

### 1.2.2 Current software modeling issues

Numerous studies highlighted that the adoption of model-based software engineering approaches is limited by various modeling-related issues. For instance, some studies point out the complexity of the modeling language, while others highlight the steep learning curve and the usability issues of modeling tools. These issues have been investigated under several aspects, to such a point that they are now acknowledged by the modeling research community.

Today, we know these multiple problems, but when we ask "*What is the first issue to tackle to help modeling practitioners?*", the answer is far from obvious. Literature provides scattered portions of information, often focused on one aspect of modeling, such as the complexity of the language, or the use of the tool. However, practitioners mix all these concepts together to achieve overall modeling tasks. This might cause unexpected issues to emerge. In the meantime, it is important to understand whether practitioners actually experience all aspect-specific problems reported in the literature.

One of the goals of this thesis is to understand which, if any, issues should be addressed in priority, according the practitioners. Thus, we put our results in perspective with the literature to potentially identify problems that may not have been addressed in the literature yet. Our approach on this point is closer to applied research rather than fundamental research, as it builds on the desires of the population being studied rather than on the theories validated in the literature. Thus, the thesis aims to help academics but also practitioners who would be interested in improving their tools, to adapt them to the field problems, identified by the practitioners and the scientific community.

### 1.2.3 A solution to software modeling issues

During three years dedicated to research in the software modeling field, we encountered many articles identifying the modeling problems, as mentioned in the previous section. This is probably the case for many readers of this manuscript and for researchers, provided that they regularly work in the modeling field. The mention of these problems is recurrent in literature, to such a point that it is now almost an accepted fact in the community that *modeling raises many issues* (but also solves many).

However, while much work has been done on identifying such issues, it seems that much less research effort has been devoted to identifying solutions to these problems. In any case, it is clear that these solutions

are much less known to the community than the problems. For example, modeling researchers can easily cite the problem of modeling tools, which appear to be a barrier to the broad adoption of MBSE approaches. However, it seems much more complicated to list concrete and proven solutions to this problem, e.g., to answer "How to improve modeling tools in a concrete way?". This observation is not specific to modeling tools, and can be applied to almost any other modeling problem identified in the literature.

The thesis aims to show that software assistants can help to compensate for the previously identified problems of modeling (see the previous section). Then, in order for these assistants to fulfill their role, and hence propose better alternatives than the currently available tooling, we particularly study *how to assist modeling*. The development of a good solution first requires the perfect understanding of both the context and the problem to be addressed. Thus, this manuscript presents an analysis of the modeling task and the problems of modeling, as a basis for understanding how to make software assistants a real solution to modeling problems. Finally, we present a set of guidelines for defining software assistants as concrete solutions to current modeling problems.

## 1.3 Research methodology

In this section, we describe the research approach that we conducted over the three years of the thesis and introduce the research questions that this manuscript addresses. Then, we present the 3-part structure of the thesis.

### 1.3.1 Research approach

The overall goal of the thesis was to investigate how to design systems capable of cognifying [37] the current modeling tools, e.g., capable of integrating knowledge into software modeling tools to assist modelers. We specifically investigated the Human-Computer Interactions between such systems and the modeling engineers more than the technical aspects such as recommendation or graph algorithms. This interest had been motivated by both the complex nature of the modeling task and the current shortcomings of modeling tools. In this thesis, we particularly emphasize the prevalence of well-designed interactions on the performance of the back-end algorithms for assistance systems for software modeling.

To do so, we followed an approach similar to the user-centred design methodology as defined in the ISO 9241-210 standard named *Human-centred design for interactive systems* [79]. This methodology of design thinking, which is now commonly used to design the User eXperience of new user-oriented systems, is based on four main stages as presented in Figure 1.1, which then can be detailed as presented in Figure 1.2. We detail these four main steps and introduce how we applied them to our research about software modeling assistants.

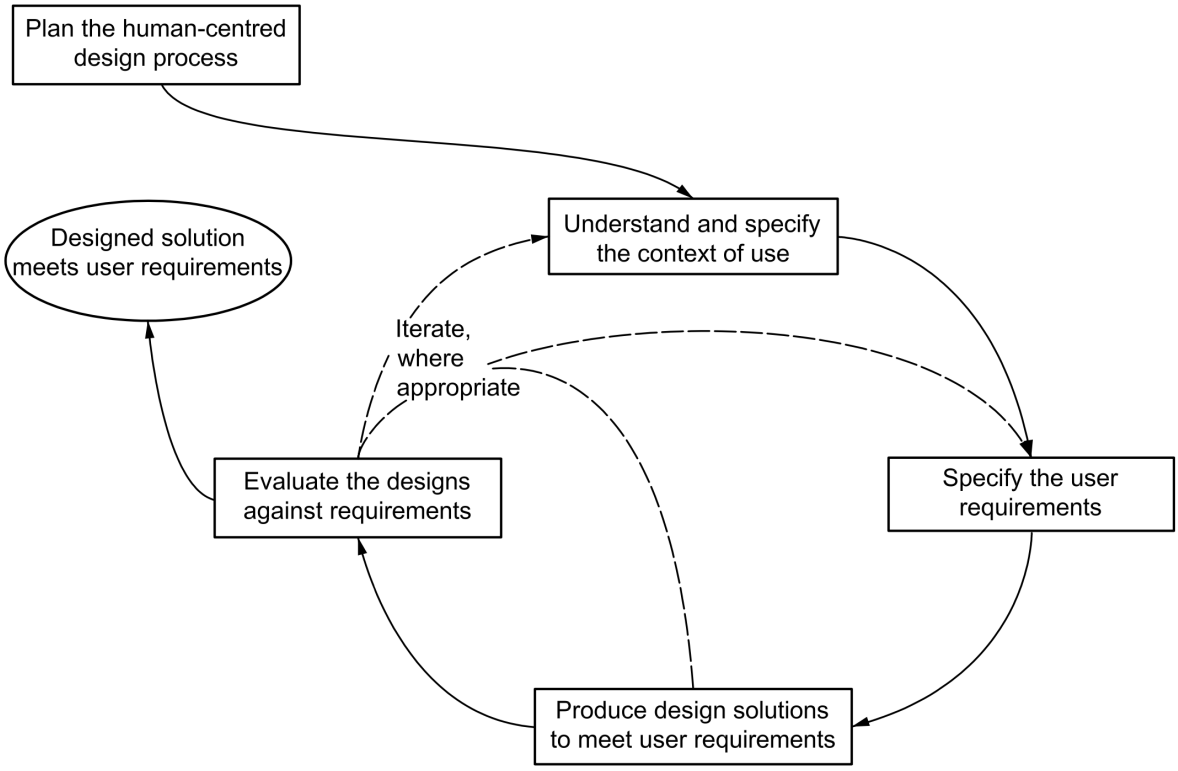



Figure 1.1 – The user-centred design iterative process from ISO 9241-210 [79]

1. **Understand and specify the context of the use.** During this step, designers are expected to conduct research to develop an understanding of the potential users of the system, and to observe where users' problems happen. This includes the understanding of (i) the users and (ii) their characteristics, the (iii) goals and tasks of the users, and (iv) the environment of the system. In our case, the previous refer to the understanding of the various (i-ii) *profiles of modeling tool user*, (iii) the *nature of the modeling task*, the *software modeling processes and inner tasks*, as well as (iv) the current *modeling tools and their issues*.

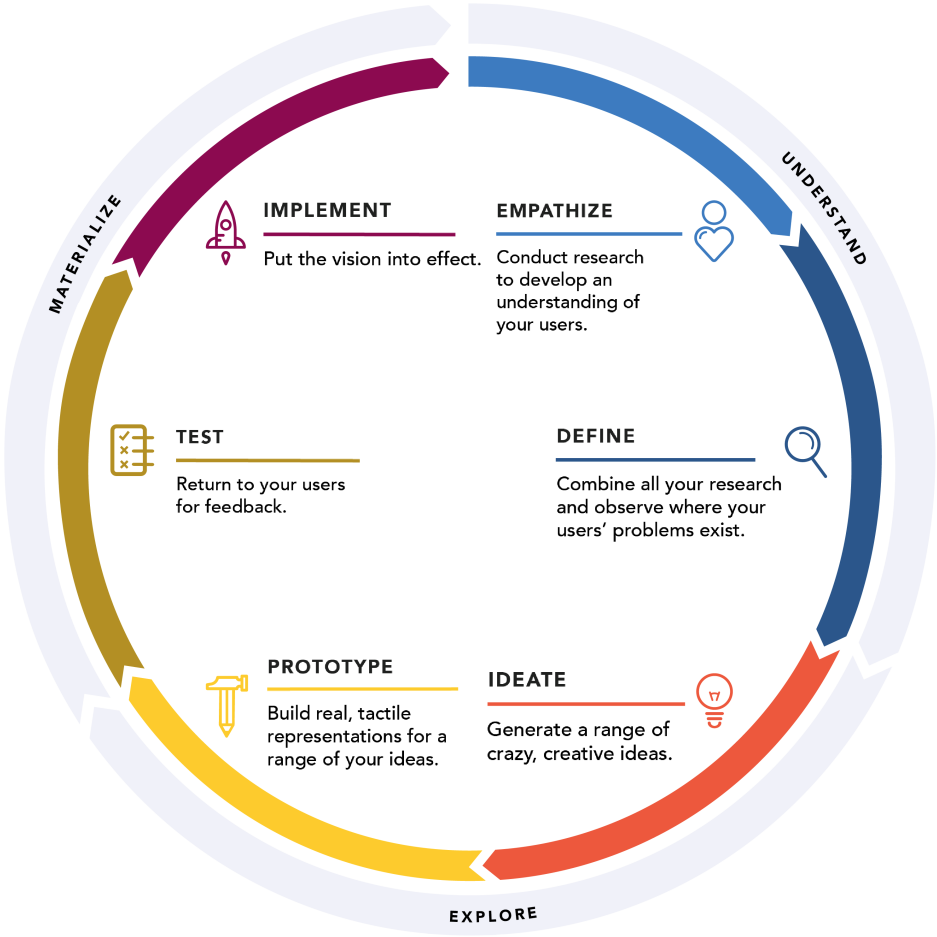



Figure 1.2 – Detailed steps of design thinking adapted from [66]

2. **Specify the user requirements.** This step mainly refers to identifying users' needs. User experience design methods include observation, surveys, or interviews, to gather the maximum information about users and their true needs. This translates to our case by gathering the modeling tool users' expectations and needs in term of assistance when performing digital modeling.

3. **Produce design solutions to meet user requirements.** It is during this step that a prototype is developed for further evaluation. The ISO standard only refers to the design of user tasks, user-system interaction, and user interface that meet the requirements identified in 2. However, it does not cover the way data is produced to be displayed by these components. In this thesis, the prototype creation instead requires a full software assistant to be developed so evaluations can be performed. It hence includes the previous components to be developed, as well as technical components such as databases or algorithms.

4. **Evaluate the designs against requirements.** During this last step, designers are expected to conduct user-centred evaluation, by performing user-based testing and/or inspection-based evaluations using usability and accessibility guidelines or requirements. These two evaluation methods are further detailed in the ISO 9241-210 standard. Thus, in order to evaluate and validate the research results of this thesis, empirical experiments with real modeling tool users should be conducted.

We followed these steps to guide our research approach about cognifying modeling tools. Our research work has thus allowed a better understanding of the modeling tasks, the users' needs, and the current modeling tool issues, which is synthesized here in this manuscript and has been partitioned into several articles. These articles underwent a partial and preliminary validation by being submitted to peer-reviewed venues, as listed at the end of the manuscript. Our research approach was then structured to be split in different parts, and to answer the overall thesis research questions presented in the following section.

### 1.3.2 Thesis research questions

To achieve our goals in understanding and assisting the software modeling activities, this thesis addresses the following research questions.

- **T.R.Q. 1.** What is a software assistant for software modeling?
- **T.R.Q. 2.** Are there software assistants for software engineering (and software modeling) in the literature?
- **T.R.Q. 3.** What are the common characteristics of software assistants for software engineering from the literature?
- **T.R.Q. 4.** Are there identifiable ways of designing software assistants for software engineering that emerge from the literature?
- **T.R.Q. 5.** Are the software assistants available in the literature in line with the expectations of modeling practitioners?
- **T.R.Q. 6.** What are the key concepts in modeling assistance?
- **T.R.Q. 7.** What are the guidelines to follow when designing software assistants for software modeling?

One of the main interests of these research questions concerns the social impact of introducing digital partners in a software engineering task. Indeed, modeling assistance has almost always been performed between colleagues, from human to human. Integrating a machine to act as a human peer in a work-related task has a potentially important social impact. The research questions investigate how to integrate these digital peers into the work process of engineers in the most suitable way. The social and emotional aspects such as users' perception about what is good or bad, their preferences, their creativity, or their propensity to trust are part of this process, and are thus addressed by these research questions. Composing a human-machine team shares many characteristics with creating human-human teams, as done everyday in companies. These issues can then be generalized to the problem of composing a successful project team from heterogeneous profiles, often addressed in software engineering.

These research questions aim at understanding how to support the modeling activity. Thus, they may support economic interests at different scales. Assisting the modeling task aims at reducing the time or effort spent on it. The reduction of these two factors is sensibly linked to the productivity increase of modeling engineers, which is of economic interest to the company that employs them. Assisting conceptual modeling can potentially improve the quality of the models produced, which in turn has a positive impact for the company. Editors and designers of software solutions to support modeling might also find a strong economic interest in the answers to these research questions. In particular, the outcomes might allow them to better understand how to design (i) a product more adapted to the target population, (ii) to produce products with better acceptability, and (iii) to improve the design methodology of their solutions.

These research questions are of interest to researchers seeking a better understanding of collaborative modeling, with a machine. Indeed, the notion of assistance implies the collaboration of two actors, whether human-human, or human-machine. In particular, we deal with the key notions of assistance, valid in both cases, and then refine our work on how to translate these key notions into software systems. Our approach is entirely based on scientific foundations coming from the literature and may therefore be replicated and potentially generalized to assistance for other tasks. The guidelines we propose allow for the design and implementation of new assistance systems, and thus offer a validation alternative, based on the comparison of existing or future solutions with software assistants.

The approach guided by these research questions aims at proposing an original and innovative solution by using software assistants to tackle software modeling problems. These problems are a clearly identified theme in the software modeling research community. Our work is anchored at the crossroads of different research themes such as Software Modeling, Human-Computer Interactions (HCI), Computer-Supported Cooperative Work (CSCW), or the community of Bots systems.

### 1.3.3 Structure of the thesis manuscript

The manuscript is organized as follows. Part I investigates the current state of Software Modeling Assistants, as to understand and specify the context of the use of assistants, and to specify the user requirements about them. Chapter 2 provides context elements to embrace the nature of the modeling task and the challenges to support it with software modeling assistants. Chapter 3 reports on our effort to provide a clear understanding of the literature about software assistants for software engineering, including software modeling. Chapter 4 presents the results of the interviews of 16 modeling experts about their need of assistance at work. Chapter 5 puts into perspective the previous results of this first part, highlighting the breaks between existing software assistants and practitioners' expectations, as well as the key notions of modeling assistance.

Part II relies on the key notions of modeling assistance presented in Part I to formalize the creation process of software modeling assistants. Based on the results of our study of the literature, our systematic literature study, and our interviews, it describes the process of defining a formal framework to design software assistants for software modeling. Chapter 6 draws upon the conclusions of Chapter 5 and identifies literature guidelines to support the key notions of modeling assistance in software systems. Chapter 7 presents the application of standard ISO/IEC/IEEE 42010 to define a formal framework for the design of software assistants for software modeling.

Part III reports on our framework validation approach. Chapter 8 describes the instantiation of the previously defined framework to design a system that assists the creation of Unified Modeling Language (UML) class diagrams inside the Papyrus modeling tool. Chapter 9 reports on the preliminary evaluation efforts that have been conducted, and describes our plan to conduct broader empirical evaluations. Chapter 10 concludes on the research questions and identifies perspectives to extend or generalize our work.

# Part I

# The current state of software modeling assistance

Chapter 2

# Supporting software modeling: context and challenges

*In this chapter, we provide a broad overview of the field of software assistants, and dig into what is the nature of the software modeling task as the first step of our human-centred research approach. Based on the state-of-the-art modeling issues and modeling purposes, we present a first effort in understanding and specifying the modeling context, in which we plan to embed software assistants. This chapter sets the scene for the thesis and identifies the first issues of the thesis, namely the problems of modeling tools and assisting modeling with respect to the nature of its inner mechanisms.*

## 2.1 Tool support in Software Engineering

Tools for programmers naturally existed since the beginning of Software Engineering around 1960. At that time, they were single tools focused on some specific tasks of the Software Engineering life cycle, mainly cumbersome to use, and often acting in isolation of each other [126]. Around 1980, the increasing complexity of the solutions to be produced as well as the better understanding of the users' needs drove the improvement of the existing Software Engineering instrumentation [22]. A new wave of systems then gradually replaced tools with more com-

prehensive functionalities gathered in *environments*, such as Integrated Programming Support Environments (IPSE) and later Software Engineering Environments (SEE). These systems fall under the emerging field of Computer-Aided Software Engineering (CASE) tools, which lay the foundation for modern-day Integrated Development Environments (IDEs). As environments improve, other issues emerge such as the need for collaboration to produce ever more complex systems, which paves the way for the Computer-Supported Cooperative Work (CSCW) community and more specifically the Collaborative Software Engineering (CSE) community [67]. The CSE community seeks to enhance environments to cope with different forms of collaboration.

During the 1990s, the *agent* research fields exploded and brought to light a new opportunity for collaboration: that with the machine acting as an autonomous system with which users (or other agents) could interact and work [83]. Some agents are refined into *intelligent agents* that are reactive, proactive, and social agents tailored for human-agent collaboration [191], and applied to support Software Engineering processes [62, 75]. However, due to the lack of computing resources and/or data to exploit, such agent-based systems never became mainstream in Software Engineering [54].

The broad Software Agent community has remained active and has branched into several sub-categories. Particularly, the notion of *conversational agent* (a.k.a. bot or chatbot – coined by Michael Mauldin in 1994) is gaining importance recently, and has quickly became a must-have, especially in the sectors of customer support or video games [56, 54]. In 2016, Storey and Zagalsky laid the foundation for research on bots in software engineering and described how bots are increasingly used to support tasks that traditionally required human intelligence [174]. It has particularly been applied to Software Engineering to create *BOTse* [173] or *DevBots* [54] (bots for Software Engineering) [134]. A consensual definition established during the BOTse Dagstuhl seminar in 2020 [173] defines bots as systems featuring at least one of the following characteristics: (i) automates one or more feature(s), (ii) performs one or more function(s) that a human may do, (iii) interacts with a human or other agents.

At ICSE'06, Boehm predicted a new kind of developer-helping systems for 2020s as "that provide feedback to developers based on domain knowledge, programming knowledge, systems engineering knowledge, or management knowledge" [22]. The description of previous bot systems is almost inline with these expectations but still lacks one essential characteristic that Boehm described as "the use of knowledge". Storey

et al. [174] identify bots embedding knowledge as one specific type of bots. Thus, knowledge appears as an inflexion point, which opens the way for the study of a specific type of system —knowledge-empowered DevBots— that we will call Software Assistants for Software Engineering.

## 2.2 A new wave of assistance systems: Software Assistants

Software assistants are progressively spreading at work, into the Software Engineering community. However, it has been several years that they made their way into our daily lives. In this section, we elaborate on the need to understand what is a software assistant for work tasks, and clarify the role of knowledge in this new wave of systems.

### 2.2.1 Digital assistance systems: Software Assistants

Digital assistance systems such as Google Home, Amazon Alexa, but also recommender systems on Netflix or Amazon, or website-integrated chatbots, have spread into our lives recently, to the point that they now seem commonplace [111, 68, 184]. These software assistants aim to improve our daily life, by performing actions for us, answering our questions, or anticipating our needs. To do so, they often offer smart interactions, which usually adapt to the conditions in which they are used [120]. However, although they are widespread at home, few of these technologies have been applied to the work environment. Indeed, it appears that research in the area of applying these assistance technologies to the world of work, and particularly to software modeling, is underdeveloped. This may be related to the insufficient understanding of the tasks performed during modeling, which is a complex creative activity aiming to solve ill-defined problems [181]. One of the objectives of this thesis is to understand how these technologies anchored in our daily life can be applied to work tasks, and in particular to the modeling task.

### 2.2.2 Knowledge provided by software assistants

To be truly useful and effective, software assistants must be able to answer users' questions or perform tasks for them. To do so, they must demonstrate an understanding of the context, but also possess the

required knowledge to achieve their goals. Like human assistants [53], software assistants must then embed *knowledge*.

Knowledge appeared in the 1980s in the scope of Software with knowledge based systems [163]. However, it is with the increasing amount of available storage and computation resources that it appeared in the 2000's as a revolution for digital systems, involving fundamental changes in the way people relate to their own knowledge [20]. *Knowledge* is a broad term which encapsulates different notions and which has no consensual definition [154]. Nevertheless, it is commonly admitted that data can lead to information which, in turn, can lead to knowledge, based on the Data, Information, Knowledge, and Wisdom (DIKW) hierarchy [3, 57]. In the scope of software systems, we adopt the definition of knowledge as (i) the result of the analysis of structured information, (ii) related to the current context, problem, or activity, (iii) and tailored to the user's needs [147, 133, 57].

Based on the highly influencial description of the DIKW hierarchy of Rowley [147], we provide the following description of data, information and knowledge in the frame of Software Engineering:

- *Data* is the content of the considered software artifacts.

- *Information* is a fact about one ore more artifacts, resulting from their simple reading.

- *Knowledge* is the result of the analysis and combination of information, valuable in the scope of the current problem, in the current context.

Thus, the notion of knowledge only makes sense when linked to a specific task or problem. Let us illustrate these concepts with an example. John codes a Java program and launches the execution in his IDE. An error occurs about a graphical element that John coded, and the error message is displayed in the console. John wants to understand what portion of the code causes the error. In this context, data is represented by all the files containing Java code as well as the error message displayed in the console. One information could be that 35 Java files contain references to the graphical element (as all information, this is not context-dependant). Then, one knowledge would be that the file causing the error might be among a shortlist of three recommended files.

## 2.3 Challenges of Software Assistants for Software Modeling

Software assistants and their knowledge-based mechanisms appear as a solution to improve the daily life and to propose new ways to interact with electronic and software environments. In the meantime, while research about them still appears limited, they seem to gain popularity in industrial software development tools. In this section, we first clarify the notion of software assistant for software engineering, and highlight why these systems might appear as a solution to support modeling engineers in their work. Then, we present how modeling assistants are positioned in relation to the existing literature on modeling issues. More precisely, we discuss why modeling assistants are a novel and relevant solution to support modeling, with respect to the literature about the actual practice of modeling and the modeling tools. Finally, we provide a detailed description of the nature of the modeling task, to clearly define what modeling assistants are to assist. This description is the foundation of our later definition of modeling assistance, as it identifies the main challenges to deal with when supporting the modeling task.

### 2.3.1 Software Assistants for Software Modeling

*Software assistants* aim to facilitate the access to knowledge through well-designed interactions, such as presented in Section 2.2. Software engineers exploit knowledge in many Software Engineering activities [151], on a wide range of software and/or domain specific topics [97]. This recurring need, in conjunction with the evolution of technologies, calls for a new wave of systems for software engineering that puts knowledge at the heart of their logic and interactions. These assistance systems are essential to enable engineers to keep control over increasingly large and complex software systems to design and maintain.

To describe these systems, we adopt the definition of software bots presented in Section 2.1, and define *Software Assistants for Software Engineering* as software bots which provide users with valuable knowledge to help them identify, understand, or solve a problem. The notion of knowledge refers to the definition introduced in Section 2.2.2. For the sake of clarity in the rest of the paper, we will refer to software assistants for software engineering with the shorter version software assistants.

In some previous works, software assistants may also be referred to as Intelligent Assistant [119], IA-based digital Assistant [104], Intelligent User Assistance System [105], Virtual Assistant [21], Smart

Assistant [116], Intelligent Agents [33], or shortly Bots [173]. While the description of these systems seems to converge on the notion of assistant, some also involve the notion of Intelligence. As knowledge appears to be only one component of what constitutes intelligence [2, 106], we refrain from qualifying software assistants of intelligent or smart. However, artificial intelligence techniques, such as machine learning or ontologies, might be embedded in software assistants to create, organize, or filter the knowledge that is required.

Software assistants may help users make a decision and eventually perform a task according to this decision with a certain degree of autonomy. Their outcomes might not be deterministic, as they adapt to each problem and context. They might be used to automate manual tasks to save time and reduce effort, but they must be able to come up with new information and ideas and that may be valuable to increase the knowledge of the user. Software assistants consist in complete and ready-to-use software systems, accessible through a user interface (to be able to provides users with valuable knowledge). Therefore, single components and algorithms are not considered as software assistants.

Although there is a wide variety of bots and software tools embedded into IDEs that perform automated tasks and are used during the software development process (e.g., refactoring, search, and indentation tools), it is worth noting that software assistants are still not mainstream.

### 2.3.2 Addressing modeling issues

Software Assistants for Software Modeling aim to assist modeling tool users by augmenting them with knowledge. However, one may wonder why software engineers need this new type of system. Indeed, the modeling research community has been working on this topic for more than forty years, and has already produced numerous modeling tools or tool extensions.

In this section, we first show that the modeling community has been very active in identifying the reality of practices in the field, concerning the modeling methodology, and the use of the UML language. In the meantime, we outline several studies that have pointed out the general problems related to modeling. In a second step, we highlight the fact that modeling tools remain limited in supporting the modeling activity, and that they can even appear as a barrier to the adoption of modeling methodologies. Finally, we justify the difficulty of tools to support the modeling task by emphasizing the complex nature of this task.

The definition of the nature of the modeling task is a central point of this manuscript, as it is at the origin of the major problems encountered

during the design of assistance systems for modeling. These observations shed light on the need for new modeling-supporting systems, tailored to the users and to the complex nature of the modeling task.

### The state of UML software modeling practice

The practice of software modeling is strongly coupled to the modeling language used. In this thesis, we limit our scope to the use of UML within software modeling tools. This choice had been guided by the numerous studies indicating that UML remains the most taught and used modeling language. Our approach being user-centred, we decided to choose UML to maximize the population concerned by our results, and to maximize the potential participants for our user-studies. This section aims to identify studies that report on the practice of software modeling, including the consequences of using UML as a modeling language.

The work from Grossman et al. [65] identified that the main objectives for using UML were to capture and communicate requirements, and guide the development of code. These objectives are extended by the works of Hutchinson et al. [77, 76] who propose a new and finer classification of tasks as the use of models for team communication, for understanding a problem at an abstract level, to capture and document designs, the use of model-to-model transformations, and use of models in testing, code generation, or model simulation. To achieve such goals, practitioners sometimes encounter issues or challenges. Ozkaya [128] focused on identifying practitioners' challenges when practicing software modeling. Their results provide a coarse-grained analysis of the categories of challenges and identify the use and learning of modeling languages as major challenges in software modeling. In their survey about the perceptions of software modeling, Forward et al. [58] highlight that practitioners face issues with modeling tools (as identified in the following section), but also with the use of the modeling language itself.

While modeling with UML could serve multiple purposes, it appears that its use is sometimes partial, informal, and not widespread in organizations. The study of Dobing and Parsons [47] highlights the fact that the use of UML modeling should not be considered as exclusive to software professionals, and that there should be a better understanding of its diagrams by all stakeholders. Dealing with the language complexity, Chaudron et al. [40] also show that developers tend not to respect the standards of the language, and that they use it rather informally. Petre's study [136] goes in the same direction and indicates that the use

of UML is most often selective (only a few elements and diagrams of the language are used) and informal. Budgen et al. [35] propose a systematic literature review investigating empirical evidences of UML aspects such as metrics, comprehension, model quality, methods and tools.

The previous studies identify issues related to the way modeling is selectively exploited in software engineering companies, most often by highly-technical profiles, while it should be generalized to many more collaborators and stakeholders. Such issues call for more support to broaden the adoption of software modeling in companies, especially to non-expert profiles. This support might include guidance on the understanding of the concepts of the modeling language, of the internal modeling methodology, or of existing diagrams. Despite the extensive research work on this topic, the literature still highlights the *need for assistance* in applying modeling methodologies and in using the modeling languages.

These problems remain to be addressed by new systems, such as software assistants for software modeling. These general modeling issues are fed, if not caused, by the tools used for modeling, as stated by Forward et al. [58]. The following section presents the literature concerning the issues of modeling tools.

### Software modeling tools

For many years, modeling tools have appeared to be an obstacle to the development and adoption of software modeling methodologies [88]. In 2013, Hutchinson et al. [190] called on researchers and industry to match tools to people, not the other way around, and to put more effort on studying processes, instead of tools. They particularly stated that very little research has been carried out on how tools can or cannot support this processes, and on how to develop simpler tools that can fit into existing processes with minimal tailoring. Our study responds to this call to identify the processes and tasks of software modeling practitioners, in order to propose systems capable of helping them.

In their paper, Badreddin et al. [12] replicate the survey of Forward et al. [58] ten years later to uncover trends of the software modeling practice. While their survey highlights some increase in the adoption of the broad practices of modeling as well as formal modeling, their results suggest a persistent dissatisfaction with software modeling tools being inadequate in their support for collaboration and communication. Thus, since 2007, the industry does not seem to have succeeded in fixing the acceptability issues of their modeling tools. This may indicate a research gap in this field, or the difficulty of industrializing the research results.

The articles from Planas et al. and Safdar et al. [137, 152] also investigated the usability of modeling tools used in industry by comparing them. Both papers asked students to reproduce diagrams using different modeling tools. Although their results allow to compare the ease of execution of a feature from one tool to another, they do not allow to identify features that might be missing in the tools.

While the previous section highlighted the need to support the spread of the modeling methodologies and the use of modeling languages, this section demonstrated that modeling tools still struggle to satisfy modeling engineers. Particularly, the unmanageable complexity of modeling tools coupled with their usability issues appear to be a barrier for engineers to perform modeling on these dedicated tools. This highlights the need for existing tools to evolve, to *support* users instead of hindering them. This is the role of software assistants for software modeling to augment existing tools to better support users. However, to not reproduce the errors from the past, it is crucial to exactly understand the task that such systems should support. More precisely, it is essential that modeling assistants focus and address the issues inherent to the very nature of the modeling task. In the following section, we detail how complex this task is, to provide a clear view of the challenges that modeling assistants shall embrace.

### 2.3.3 The nature of the modeling task

In previous sections, we showed that, despite their ability to support various tasks and activities, software assistants have still not clearly made their way into the support of software modeling. We also identified that modeling is still facing issues that appeared years ago about the methodologies, the languages, or the tooling, and that are still currently relevant. Modeling assistants could then be a solution to address these problems, by augmenting the working environment of the users and then facilitating the access to modeling features or knowledge. However, in order to demonstrate an added value compared to the existing tooling, these systems shall thoroughly meet the needs of the user, but especially meet those of the modeling task. Thus, in order to best support the modeling task, we need to study in more detail the processes that take place when we practice software modeling. In this section, we clarify our definition of software modeling, and describe its problem-solving nature, its creative nature, the ill-defined aspect of the problem it raises, and highlight the risks and constraints it faces.

### Software design and software modeling

According to the Object Management Group (OMG)[1], (software) modeling is the designing of software applications before coding. This definition, that we admit, merges two processes that we will differentiate and explain in this subsection, *designing* and *modeling*.

In his thesis about understanding and supporting software design in Model-Based Software Engineering (MBSE), Jolak [84] defines the process of designing as *"the process of thinking about, pondering over, making, shaping, and evaluating design decisions for something that is to be created"*. Budgen [34] identifies different actions that take part during the process of designing:

- identification of the design actions to be performed;
- use of the representation forms;
- procedures for making transformations between representations;
- quality measures to aid in making choices;
- identification of particular constraints to be considered;
- verification/validation operations.

All these subtasks aim to structure the mental model of software engineers, by refining their perception of the problem environment, organizing it, and refining their mental picture of the design based on their mental picture of the context, as described by Ralph [144]. Then, as stated by Budgen [34], software engineers can start producing a *design*, which is a set of rules that describes how the solution should be built.

At some point, to be exploited, the envisioned design has to be produced on a media, to be communicated, for refinement or sharing. In order to communicate this design, software engineers enter the process of *modeling*. Modeling is a way to externalize or express the software design decisions that are stored in the mind of software engineers. However, as the whole model might not be depictable on one single diagram, choices must be made on the diagrams to create and the content to include. This leads to Jolak's [84] definition of the modeling process as *"the process of creating systematic and abstract representations of a concept by choosing what to represent and how to represent it"*.

These two processes are iterative and intertwined, so that it is difficult to consider them separately. Thus, in this thesis, the notion of *software*

[1] https://www.uml.org/what-is-uml.htm

*modeling* refers to these intertwined design and modeling processes. The representation of a part of a mental model as a model artifact allows for creation of a partial solution, which fosters the refinement of the problem, and leads to new problems that are then considered. This is the very principle of problem-solving activities that we present in the following section.

### A creative problem-solving task

The purpose of software modeling is to produce a solution to a problem [34]. This problem is typically represented by a set of constraints which includes financial, technical or managerial constraints. Then, it is the software designers' work to provide a description of how these constraints and users' needs are met with their solution. The software modeling process is therefore essentially a problem-solving task. It involves the designer in evaluating different options, and in making choices using decision criteria that may be complex [34].

This constant decision-making and problem-solving process is already expressed by Fox and Kessler [60] in 1967 who state that "*obviously, the designer of a software support system faced with these impossible objectives finds his life to be one of constant decision making as he comes up with the compromises that give the best possible approach to this solution.*" This problem-solving task is further complicated by the fact that the modeling activity consists of a creative activity, as stated by Budgen [34], which calls for *new* and *adapted* solutions to the problem [45, 25]. Thus, the nature of the problem to solve then completely changes, and falls into the category of *ill-structured problems*.

### Ill-structured problems

Bonnardel and Zenasni [25] state that from a cognitive point of view, a main characteristic of creative design activities is that the initial state of the problem is *ill-structured*, which applies to software modeling. At the initial state, such problems cannot be completely circumscribed, because their deep understanding requires several iterations between the analysis of the problem and the proposal of partial solutions. Then, it is only through during the problem-solving process that software engineers refine their mental representation of both problem and solution spaces iteratively and simultaneously, in a process called *co-evolution* [48]. Budgen refers to these problems as *wicked problems* [34].

Form the work of [170, 64, 63], Darses et al. [45] identified the characteristics of ill-structured problems, which precisely apply to software

modeling, as follows:

- Problems tend to be large and complex. By this we mean that they are not generally confined to local problems, and that the variables and their interrelationships are too numerous to be broken down into independent subsystems.
- A consequence of this complexity is that the resolution of these problems requires the pooling of multiple skills, which necessitates collaborations within the same team.
- The solutions to a design problem are more or less acceptable: there is no single *right* solution.
- We cannot distinguish two consecutive phases: analysis of the problem, then resolution of this problem. The two phases interact: there is no *the* problem that precedes *the* solution.
- There is no predetermined design resolution path: one knows a certain number of useful procedures and methodologies, one can rely on similar projects already dealt with or on existing prototypes, but each time one has to recombine, if not reinvent, strategies to elaborate a solution.
- The evaluation of solutions is difficult to achieve other than through mental simulation, on the basis of graphic representations or models that often poorly reflect reality. It is limited by the fact that the generation of all possible solutions is impractical and that there is no objective metric (set of criteria). As it is often deferred to the establishment of the final solution, the possible questioning of design choices can be very costly.

**The ill-structured nature of the software modeling problem completely conditions the work presented in this thesis. As there is no *single* solution, and there is no *one* method to create the *perfect* software design, assistance systems for software modeling–and more generally for creative problem-solving tasks– do not compete in the race for certification, nor for the quest of maximum precision, which animates the current context of artificial intelligence. Instead, these systems should support the user in identifying partial solutions, sometimes incorrect solutions, which are the key to the better understanding of the problem space, and finally lead to the design of an acceptable solution.**

### Risks and constraints

In 1970, J. Christopher Jones, one of the pioneering design methodologists, stated the following about software design [85]. *"The fundamental problem is that designers are obliged to use current information to predict a future state that will not come about unless their predictions are correct. The final outcome of designing has to be assumed before the means of achieving it can be explored: the designers have to work backwards in time from an assumed effect upon the world to the beginning of a chain of events that will bring the effect about."* This quote perfectly highlights the main risk of software engineers to fail in making the right design decisions at the right time, and then direct the project towards failure when it comes to software construction or testing. Current software design methodologies such as the agile method, allow for more flexibility and thus, the opportunity to fix some of the bad decisions before it grows irreversible. However, these methodologies have a very little effect on the design of systems that are to evolve over time, and whose bad initial design may prevent or greatly impact the development of future functionality.

Risks may also appear more easily when conceiving critical systems, for which any design decision must be explainable and traceable. In these cases more than ever, the system designers must respect the standards imposed by the specification agencies, by the companies, by the teams, in addition to adapting to the project constraints. These constraints can be budgetary, human, or technical and come from the software engineer's company in charge of building the solution. The customer's needs provide specifications that the solution must meet, and also add more constraints to match.

These constraints and risks add additional pressure on the software developer in charge of making design decisions. Thus, to support the software modeling process, assistance systems must take risk and constraints into account, and inform users of their competence to assist them in these complex tasks.

## 2.4 Conclusion

This chapter has clarified our theoretical vision of software assistants for software engineering and for software modeling. We first defined software assistants, a new type of software systems that aim to assist users through the use of knowledge and well-designed interactions.

Then, to determine if software assistants could be appropriate solutions to tackle modeling issues, we provided a detailed description of software assistants for software modeling and confronted them to the literature about software modeling issues.

We highlighted that the literature depicts a strong need for more support in broadening the adoption on modeling methodologies, the mastery of modeling languages, and the use of modeling tools. All previous issues relate to *users* and *knowledge*, whether it is about modeling knowledge, language syntax knowledge, or tool-related knowledge or expertise, which qualifies modeling assistants as ideal solutions to address the modeling issues identified in the existing literature.

Finally, we provided a detailed description of the nature of the modeling task. This enabled to precisely identify the challenges that software assistants shall embrace to qualify as a solution for *modeling* assistance. We claim that the nature of the modeling task totally conditions the way that assistance should be provided for the modeling task, due to its creative, risky, and ill-structured characteristics.

As software assistants appear as potential solutions to support software modeling, we investigated the existing literature about software assistance. We analyzed the Software Engineering literature to identify their common design characteristics in order to better understand how to build such systems. As stated in this chapter, the research effort about applying digital assistance technologies to software engineering appears limited. Thus, in order to build a clear understanding of the research landscape on software assistants for software engineering (including software modeling), we have carried out a systematic mapping study, to have a reliable picture of the literature. This work is presented in the following chapter.

Chapter 3

# Software Assistants for software engineering in literature

*This chapter reports on a Systematic Mapping Study (SMS) on software assistants for software engineering, which led to a journal article that has been submitted and is currently under review. We conducted the full protocol of this systematic mapping study, and our journal article was written with Loli Burgueño, and reviewed by Xavier Le Pallec, Sébastien Gérard, and Jordi Cabot. This work contributes to our effort in understanding the current context, e.g., the current state of the literature about software assistants, as part of our human-centered research approach. Results indicate that the modeling-related tasks do not have sufficient support in the literature, and draw a global overview of the research landscape about software assistants for software engineering.*

## 3.1 Related works

This section positions our work with respect to systematic literature reviews, mapping studies and surveys on topics related to software assistants and software engineering.

To the best of our knowledge, there is no study with the same focus as ours. On the contrary, the available studies either focus on one specific kind of assistant-related approach and study its application in

the Software Engineering life cycle, or focus on one of its stages and identify the assistant-related approaches and systems. Let us list and describe them below.

Some related works aim to investigate the use of one specific approach during the various stages of Software Engineering. For instance, Gasparic et al. [61] conducted a systematic literature review on Recommender Systems for Software Engineering (RSSE). They identified 47 implemented systems in papers published before 2013, and analyzed their inputs, their outputs, the effort required from engineers, and the benefits they provide to their users. Their results showed that RSSE mainly supports reuse, debugging, implementation, and maintenance phases/activities, which we also cover in this study (see our inclusion criteria in section 3.2.2). They found that most RSSE mainly recommend source code, and only some of them digital documents. Our work differs from theirs in several aspects. Firstly, the RSSE that the authors have studied qualify as one of the types of software assistants that we consider in this systematic mapping study – as long as they present at least a fully implemented prototype of the assistant. Secondly, our research questions are broader. For instance, we elaborate on aspects such as the nature and the environment of the software assistant which both condition how the knowledge should be presented. Finally, our inclusion time frame from 2010 to 2020 updates the overview of the trends of RSSE.

In 2021, Almonte et al. conducted a systematic mapping study [7] about recommender systems in Model-Driven Engineering (MDE). The study targets the MDE tasks that might be subject to recommendations, the applicable recommendation techniques and how recommender systems are evaluated. On the contrary, our work broadens the scope of this study, focusing not only on recommender systems for MDE (which is a subfield of Software Engineering), but on fully implemented software assistants for software design and construction. Savchenko et al. carried out a systematic mapping study [158] about Smart Tools in Software Engineering with the goal to answer the question "how could the technological innovations affect the software development ecosystems and software processes?", and studied the state of the art between 2015 and 2019. While the authors try to disclose the impact of software innovations in businesses, our work puts the emphasis on the software assistants themselves (i.e., how they are built, with which tasks they help engineers, how users interact with software assistants, etc.).

Different studies also investigated the way Machine Learning (ML) has been applied in Software Engineering. Borges et al. [26] collected

177 studies from 1992 to 2019 in two groups: (i) Software Quality and Software Engineering Management (40 papers) and (ii) Software Quality and Software Test (15 papers). They conclude that Software Quality is the most frequent Software Engineering Body of Knowledge (SWEBOK) knowledge area target for both clusters (51%) and that ML is mainly used to make predictions in Software Engineering. A similar study by Shafiq et al. [164] shows that based on 227 articles about ML for Software Engineering from 1991 to 2019, 21 were focusing on requirements, 39 on architecture and design, 21 on implementation, 119 on quality assurance and analytics and 9 on maintenance. Our study is richer and broader in the sense that we do not only focus on assistants empowered with ML techniques, and more specific in the sense that we are only interested in working assistants and ignore theoretical and unimplemented approaches.

Other mapping studies focused on the use of tools during one specific task of Software Engineering. For instance, Iung et al. [81] reported on the tools to enable domain-specific language development. Sebastian et al. [161] investigated code generation using model-driven architecture, and identified implemented systems enabling code generation. Similarly, Brunschwig et al. [32] provided an overview of tools to support software modeling on mobile devices.

To the best of our knowledge, there is no previous systematic mapping study on understanding and classifying implemented and ready-to-use software assistants to support software development tasks without restricting the techniques used to achieve this goal. With this work, we aim to fill the gap and summarize implemented Software Assistants for Software Engineering systems that were presented in scientific venues.

## 3.2 Research Method

This study follows the guidelines proposed by Petersen et al. [135] to conduct Systematic Mapping Studies (SMS). A pilot study was conducted on a small number of articles to assess the suitability of the criteria and the method, which led to discussions and updates on the protocol. This section describes the different steps of the exploration phase as defined in the final version of the protocol.

### 3.2.1 Research questions

In this section, we define the research questions that drive this systematic mapping study, and provide details about our intention behind each one

of them.

- *RQ1: What are the tasks that software assistants help users achieve, in which environments do they operate and which languages do they support?* With this research question, we aim to identify which parts of the software design, construction and maintenance are best supported by assistants, as well as trends in the supported environments and languages.
- *RQ2: How do software assistants assist users?* This question aims to identify the different types of assistants (as perceived by their users) and the way they present information to the user.
- *RQ3: What kind of software technologies are used to embed knowledge in software assistants?* This question investigates how software assistants are equipped with knowledge to support users. This includes analysing their data usage as well as finding whether they exploit Machine Learning (ML) techniques.
- *RQ4: To what extend are software assistants automated?* To answer this question, we modeled interaction schemes for each software assistant in our set of primary studies in order to assess their level of automation and their trigger.

### 3.2.2 Inclusion and exclusion criteria

The inclusion criteria define the scope of our systematic mapping study and enable us to identify papers which will help build answers to the research questions. We focus on those papers that:

1. are written in English, peer-reviewed, and published between January 2010 and July 2020[1],
2. focus on at least one of the these knowledge areas from the SWEBOK [27]:
    - *Software Design subject*, which relates to creating and checking software designs.
    - *Software Construction subject*, which includes program editors, compilers and code generators, interpreters and debuggers.

[1] Motivated by the background provided in Chapter 2, e.g., the recent emergence of bots, and knowledge-engineering technologies. Software assistants were expected for 2020, so we study the last 10 years to evaluate their evolution.

- *Software Maintenance subject*, which covers artifacts visualization and re-engineering.
- *Socio-Cultural systems* for software engineering, which cover socio-cultural aspects such as social networking as identified in [140].

3. focus on the assistance of the population involved in the four processes in the previous item.

Exclusion criteria enabled us to filter out irrelevant papers. Papers featuring one of the following characteristics were excluded:

- does not provide assistance for at least one of the considered software engineering tasks;
- does not introduce an implemented and ready to use software assistant—i.e., we discarded papers that only introduce a new algorithm or technique that according to the authors *could* be integrated into a software assistant, but it does not provide a software assistant in itself;
- does not claim to have, describe or provide screenshots of the user interface;
- is a survey, a systematic literature review, or a mapping study.

### 3.2.3 Search Process and Paper selection

The search process was conducted automatically by querying the *DBLP computer science bibliography*[2] website through its Application Programming Interface (API) with a custom script. Our algorithm searched for each keyword in Table 3.3 present in the title of papers that belong to any of the conference proceedings of Table 3.1 and journals of Table 3.2. The keyword was built to cover the notion of *assistance* and its synonyms for Software Engineering. This resulted in 377 queries (29 venues × 13 keywords) to the API, formatted as follows: `KEYWORD+venue:VENUE:`. An index of venue codes for the API and the full list of queries are available online on our website[156].

The venues were collectively selected by the five authors, taking into account conferences and journals based on their peer-reviewing process and their themes about software engineering, software assistance, or both. Keywords were chosen in a similar manner to convey the notion

[2] `https://dblp.org`

of assistance. Some of the keywords take advantage of the default completion feature provided by the DBLP's API. For instance, the use of *facilitat* enabled us to retrieve titles containing words like *facilitate*, *facilitating* or *facilitator*.

The automated search was performed on the 31st of July 2020 on the DBLP's API and retrieved 4,621 items. At the time of the query, we checked that each venue listed in Table 3.1 and Table 3.2 was indexed and that it featured at least one record for the last ten years.

Table 3.1 – Selected conferences

| Acronym | Name |
|---|---|
| CHI | Conference on Human Factors in Computing Systems |
| ICSE | International Conference on Software Engineering |
| ASE | International Conference on Automated Software Engineering |
| SAC | Symposium On Applied Computing |
| TOOLS | Technology of Object-Oriented Languages and Systems |
| ESEC/FSE | Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering |
| CAiSE | International Conference on Advanced Information Systems Engineering |
| IUI | International Conference on Intelligent User Interfaces |
| PETRA | Pervasive Technologies Related to Assistive Environments |
| AAMAS | International Conference On Autonomous Agents and Multi-Agent Systems |
| CSCWD | International Conference on Computer Supported Cooperative Work in Design |
| ER | International Conference on Conceptual Modeling |
| MoDELS | International Conference on Model Driven Engineering Languages and Systems |
| IJCAI | International Joint Conference on Artificial Intelligence |
| ECOOP | European Conference on Object-Oriented Programming |
| OOPSLA | Object-Oriented Programming, Systems, Languages & Applications |

Table 3.2 – Selected journals

| Editor | Name |
|---|---|
| ACM | Transactions on Software Engineering and Methodology |
| Elsevier | Science of Computer Programmin |
| Elsevier | Electronic Notes in Theoretical Computer Science |
| Elsevier | Data and Knowledge Engineering |
| Elsevier | Journal of Systems and Software |
| Elsevier | Information and Software Technology |
| Springer | Journal of Intelligent Information Systems |
| Springer | Software and Systems Modeling |
| IEEE | Transactions on Software Engineering |
| IEEE | Software |
| PeerJ | Computer Science |

We observed that the resulting list of papers contained a substantial number of articles from socio-medical disciplines. To avoid these, we identified the set of keywords frequently used in socio-medical disci-

Table 3.3 – Search keywords

| **Search keywords** |
|---|
| assist, recommend, help, facilitat, enhanc, answer, empower, augment, aid, suggest, repair, fix, support |

plines and not in software engineering. These are captured in Table 3.4, which also includes keywords derived from the exclusion criteria mentioned in Section 3.2.2. We used these keywords to prune the list of papers by automatically discarding those whose titles contained any of these keywords. This automatic pruning phase removed 1,235 articles and left 3,386 to be processed manually by the authors.

Table 3.4 – Pruning keywords

| **Pruning keywords** |
|---|
| systematic literature review, mapping study, survey, elderly, health, medical, autism, clinical, desease, home, impairment, older, living, assistive, deaf, colorblind, disabilities, medication |

As part of our protocol, all the authors defined and agreed on the fact that papers should be discarded or kept according to the decision diagram presented in Figure 3.1.

Before manually checking all the papers, and in order to ensure that the criteria was understood equally by the authors and did not lead to subjective interpretations, two authors used the defined criteria and, supported by the algorithm in Fig. 3.1, they worked independently to perform the exclusion/inclusion phase on a subset of 10% of all these 3,386 articles, i.e., on 338 papers.

Then, we computed the Inter-Rater Reliability (IRR) using a two-way mixed, single score ICC(C,1) check for consistency. We obtained $ICC(C,1) = 0.868$ (95%-Confidence Interval: $0.838 < ICC(C,1) < 0.894$) which indicates a good reliability [90], and therefore the same understanding of the criteria defined.

Finally, one of these two authors performed the exclusion of the rest of the papers, applying the algorithm of Fig. 3.1 to the reading of the title, the abstract and finally the full paper. It resulted on the exclusion of 3,224 papers based on their title, 88 papers based on their abstract, and 52 papers based on their full content.

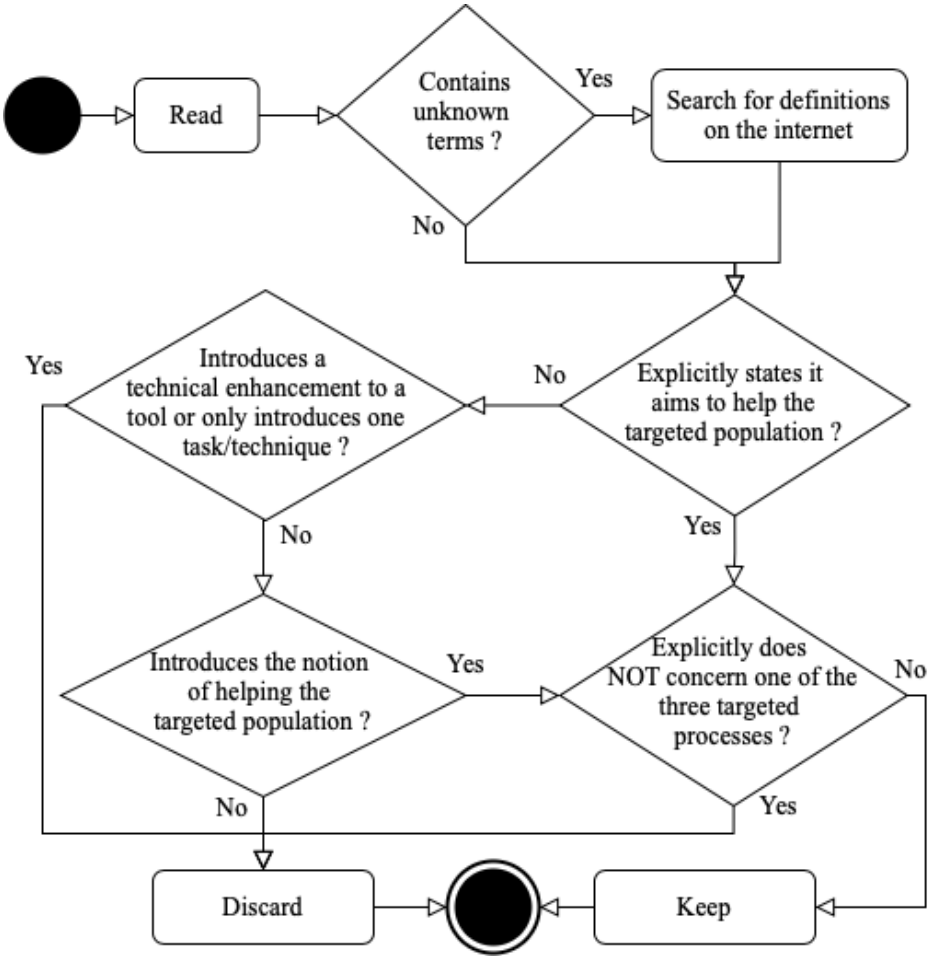



Figure 3.1 – Inclusion-exclusion decision algorithm specification for the reading of title, abstract, and full paper.

### 3.2.4 Snowballing

Those papers that passed both inclusion and exclusion criteria were exploited during the snowballing step. The stage consisted in reading the articles, especially their related work section, to identify and add to our corpus candidate software assistants that had either not been obtained during our search or that were discarded by mistake in the exclusion process. The snowballing phase mitigated the threat to validity concerning the absence of a conference or a journal in the initial venues list (Table 3.1 and Table 3.2) as it gathered new papers regardless of their origin. Each newly selected article was manually tested against inclusion and exclusion criteria, to maintain the consistency of our corpus.

We performed the snowballing phase twice. Once on the original set of papers, and then again on the papers discovered during the first snowballing phase. We did not carry out a third snowballing phase because we observed a large number of articles already overlapping

previously identified articles.

These two snowballing steps led to the gathering of 15 and 10 new articles, respectively, increasing the final number of retained articles to 47, to which we will refer as *primary studies*. Fig. 3.2 summarizes the whole search and assessment process.

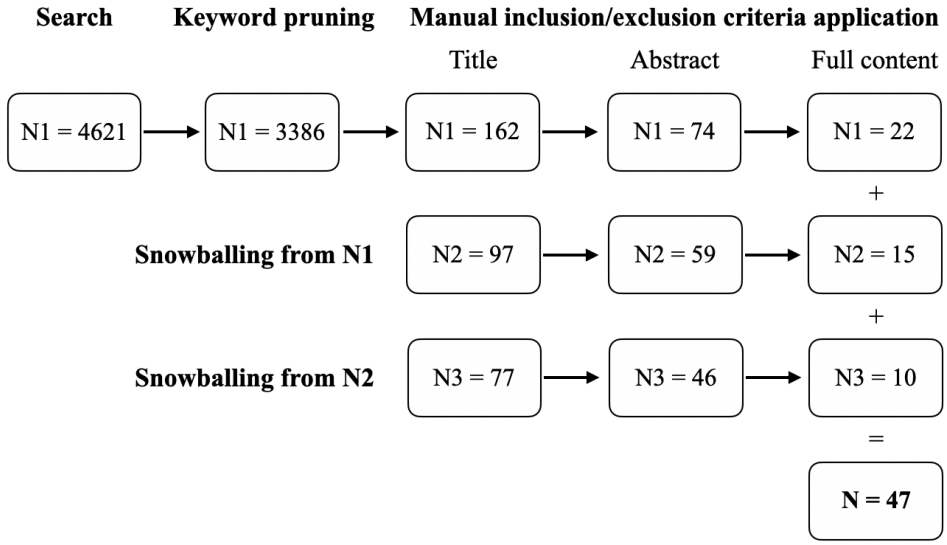



Figure 3.2 – Exploration results.

### 3.2.5 Data extraction

The data we extracted from each article is:

1. the source (journal or conference), the publication year, the title and authors;
2. the name of the tool, if provided;
3. the supported language(s);
4. the execution environment of the assistant;
5. a summary of the description and goal of the assistant as provided by the authors;
6. the datasources exploited by the assistant;
7. whether the assistant is a Recommender System (RS)—i.e., it provides recommendations—for software engineering or not;
8. if the assistant is a RS: the nature of the output, the explanation system, the confidence indicator, and the feedback system;

9. whether the assistant uses machine learning;

10. the automation levels of the assistant;

11. the high-level user-assistant interaction scheme;

12. whether a user study had been conducted;

13. whether a replication package was provided for the evaluation;

14. whether the source code of the assistant was provided.

Elements 1, 2, 5, 9, 12, 13, and 14 can be directly extracted from the paper. Elements 3, 4, and 6 refer to the actual contribution presented in the paper at the time it was written, and does not take future work and directions into account (i.e., if the tool works for Java but the paper says that it could also work for C or that it is extensible to C, we do not consider C).

The concept of recommender system for software engineering used in 7 is based on the definition adopted by Robillard et al. in their book [146]: *[...] a software application that provides information items estimated to be valuable for a software engineering task in a given context*. Elements in 8 relate to the way the RS supports human cognition to achieve certain tasks, hence we study (i) the nature of the output, i.e., whether the RS presents the information *textually*, *graphically*, or *both* [112]; (ii) the explanation system, i.e., the means with which it explains why an item is recommended (if any) [141]; (iii) the confidence indicator, i.e., how it shows how confident it is about a recommendation (if any) [124]; and (iv) the feedback system, i.e., the way it enables users to provide feedback about a recommendation (if any) [124].

Element 10 relies on the Parasuraman and Sheridan [130] framework for evaluating the automation levels of a system. They propose a model based on the four stages of human information processing to analyze the automation of a system over four different aspects: *information acquisition*, *information analysis*, *decision selection*, and *action implementation*. In order to measure the automation level for each of these four steps, we apply the 10-levels automation scale to each step as suggested in [130]. For 11, we inferred the interaction schemas between the assistant and the user as UML activity diagrams from the tool screenshots provided in the papers and the description of the User Interface (UI). In each diagram, we distinguish activities according to one of the four automation aspects they belong to.

## 3.3 Results: Analysis and classification of software assistants

This section presents the results of the data analysis we conducted on the dataset of primary studies, which are also available online [156].

### 3.3.1 Selected papers

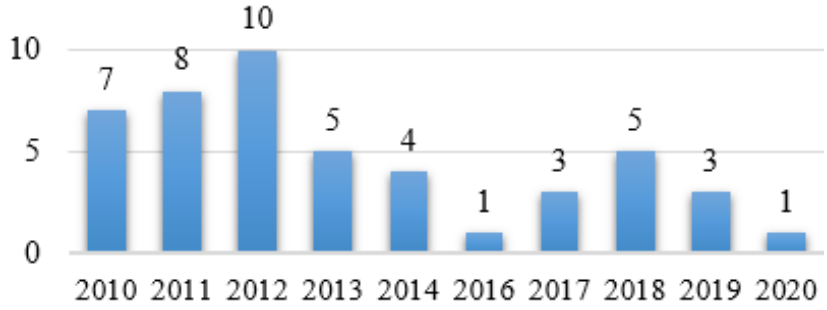
This section gathers statistics about the selected papers. Figure 3.3 illustrates the number of publications on software assistants published between 2010 and July 2020 (month in which we performed our search). On the left, it shows the variation in the number per year. We can observe that the general tendency goes down over time, reaching a maximum in 2012 with 10 papers and a minimum in 2016 with only one paper published. Figure 3.3b shows the venues in which our selected papers have been published. It is worth noting that there is no dedicated venue for the topic of software assistants and that these papers have been published in general SE conferences. It is also interesting to see how most of these papers were published in the International Conference on Software Engineering (ICSE) followed by the Conference on Human Factors in Computing Systems (CHI), which are both very prestigious conferences.

### 3.3.2 Analysis and classification results

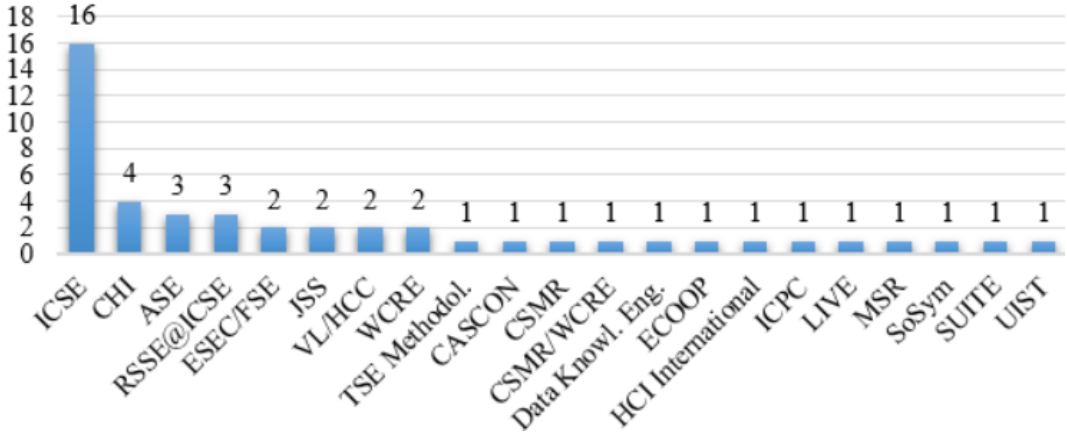
This section is devoted to answer our research questions (cf. Section 3.2.1).

**RQ1: What are the tasks that the assistants help their users achieve, in which environments do they operate and which languages do they support?**

To answer this research question, we relied on the description and goal of the assistant that we extracted from each paper as provided by its authors. From these details, we obtained information about the tasks that each assistant supports and identified the 27 tasks are listed in the second column of Table 3.5. Then, we grouped those tasks into 12 coarse-grained categories to capture their purpose. These purposes are: API/code search, Code completion & Recommendation, Code Metrics, Code Visualization & Understanding, Command Recommendation, Find Collaborators, Interface Prototyping, Modeling, Refactoring, Fix & Repair, Resource Identification, and Version Control System (VCS).




(a) Number of publications per year and type




(b) Number of publications per venue

Figure 3.3 – Number of publications

Figure 3.4 presents the number of assistants dedicated to each purpose. We can observe how some important efforts have been put into the creation of assistants for API/Code search, Code visualization & Understanding and Fix & Repair, while only isolated works have published assistants for purposes such as Interface Prototyping.

Among the wide variety of tasks, looking at Table 3.5, we can highlight that the Recommendation of Code Blocks from Queries is the most popular (supported by 6 assistants), followed by assistants for Enhancement of Default Code Completion Systems, Code Augmentation with Indicators, and the Suggestion of Code Fixes (supported by 4 assistants each). It is also worth noting how textual languages (i.e., coding) are very well supported across the whole development process—covering the tasks of finding code ideas and code excerpts, writing, refactoring and debugging/fixing as well as providing metrics— while graphical and modeling languages are under-represented.

We also extracted from each paper the environment in which the assistants work and the language (or syntax) that they support. The results can be found in the third and fourth columns of Table 3.5 and in Fig. 3.5. A star (*) means that the assistant supports all languages

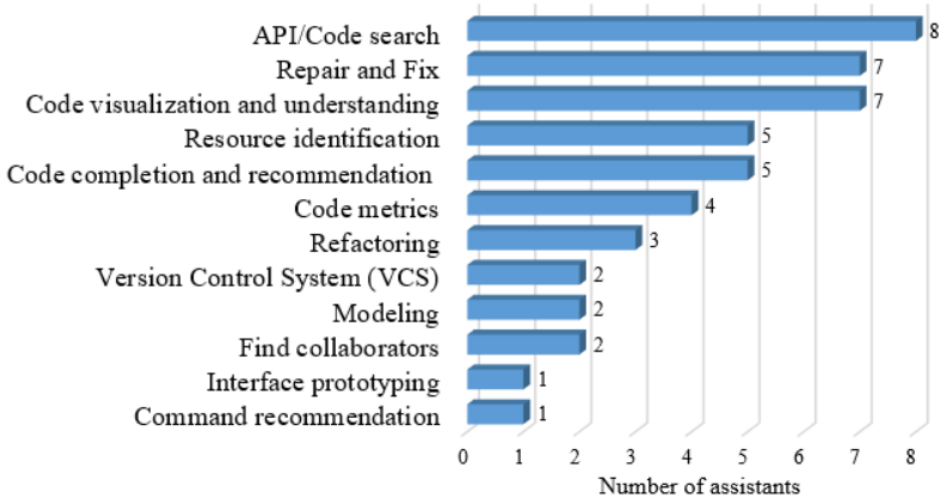



Figure 3.4 – Number of assistants per purpose

supported by the IDE. The most popular environment by far is the Eclipse IDE with 23 out of the 47 assistants. In fact, out of all the assistants, 30 of them (64%) are part of an IDE. The second most popular environment is the Web Browser (12 out of 47 – 26%) and standalone applications (4 out of 51 – 9%). There is no clear correlation between the environments and the tasks towards which the assistants help.

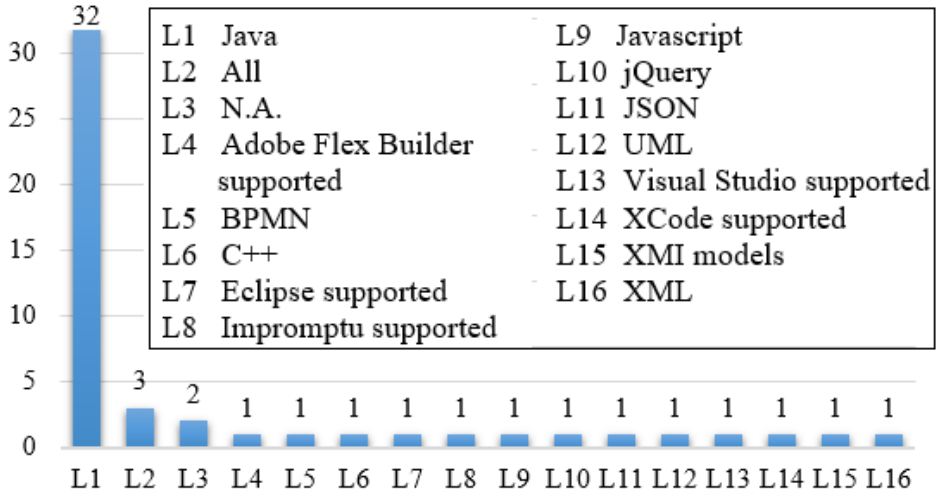
With respect to the languages supported, Java has a strong monopoly with 32 out of the 47 studied assistants (68%). It is worth noting that only 3 assistants support the top-3 *Programming, Scripting, and Markup Languages* according to the Stack Overflow 2020 survey of *Most Popular Technologies*[3], which are JavaScript, HTML/CSS, and SQL; and only two assistants support modeling languages (BPMN and XMI models).

**RQ2: How do software assistants assist users?**

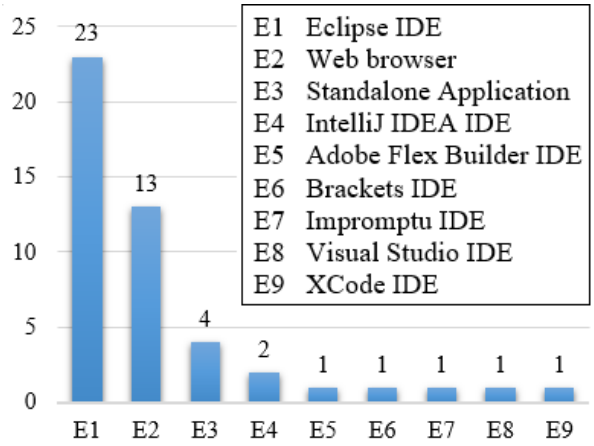
To answer this research question, we have identified the different *types* of assistants as perceived by its users and have studied, for each assistant, three Human-Computer Interactions (HCI) indicators, and the nature of the output that they provide.

**Types of assistants** We have identified that there are three main actions that assistants perform internally. The first one—and the one that every assistant integrates—is to analyze information and display the result of the analysis. The second is to help the user make a decision by suggesting one or several alternatives. The third is to perform an action

[3] https://insights.stackoverflow.com/survey/2020#most-popular-technologies




(a) Number of assistants per language




(b) Number of assistants per environment

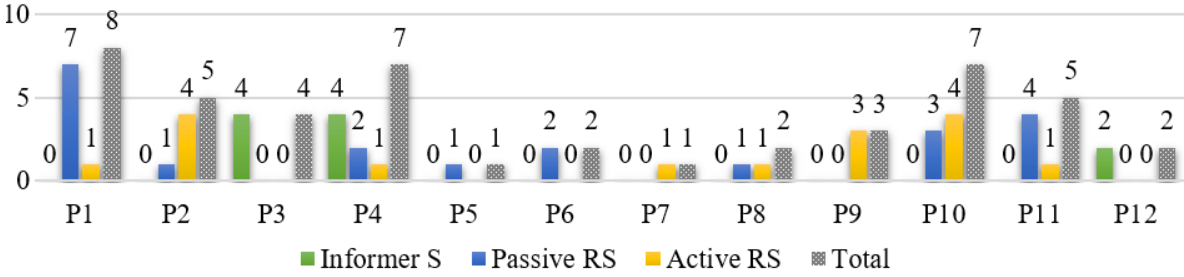
Figure 3.5 – Number of publications

based on a decision when required. Based on these three actions, we have classified the assistants and have obtained three different types as presented in Table 3.6. They are:

- **Informer System**, which helps towards reaching awareness about the work in progress or the environment. It takes raw data and information as input, analyze and/or aggregate it, and display the results without any side effect.
- **Passive Recommender System** (Passive RS), whose aim is to help the user make a decision during a software engineering task. To be able to provide meaningful potential decisions, it takes raw data or information as input, process and analyze the inputs, and eventually produce one or several alternatives for the current decision-making problem.

- **Active Recommender System** (Active RS), which extends the Passive RS by enabling the assistant to perform or implement the result of the decision.

Figure 3.6 shows the number of assistants of each type grouped by purpose.



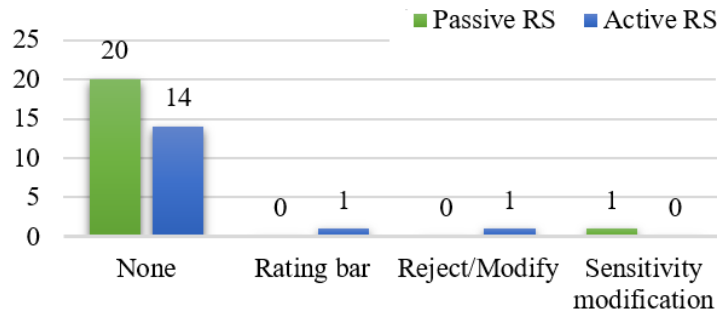

| | | | | | |
|---|---|---|---|---|---|
| P1 | API/Code search | P4 | Code visualization & understanding | P8 | Modeling |
| P2 | Code completion & recommendation | P5 | Command recommendation | P9 | Refactoring |
| | | P6 | Find collaborators | P10 | Repair & Fix |
| P3 | Code metrics | P7 | Interface prototyping | P11 | Resource identification |
| | | | | P12 | Version Control System |

Figure 3.6 – Number of of assistant types for identified purposes

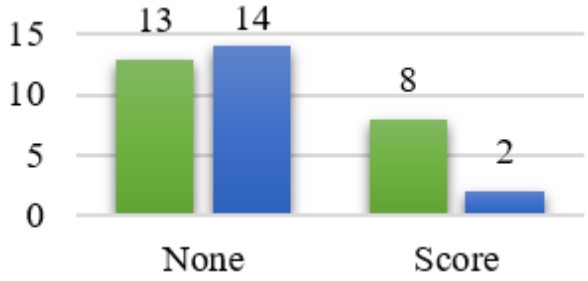
We have observed that only 21% of the assistants are informer systems (10 out of 47), 45% of them are Passive RS (21 out of 47), and 16% are Active RS (16 out of 47). This means that almost half of the assistants are Passive Recommender Systems, i.e., assistants that help make decision but do not offer the possibility to implement it. Correlating this with the set of purposes of assistants that we have identified, we can observe that all existing assistants for Code Metrics are of type Informer, while there are no informer systems for purposes such as API/Code search, Repair & Fix, Code completion & Recommendation and Resources Identification. This has some logic given that in these categories, the tasks that the assistants perform are likely to require making decisions and, in occasions, taking actions, too.

**Human-Computer Interactions (HCI) indicators** The communication between informer systems and users is unidirectional and users are simply consumers. Unlike informer systems, both passive and active recommender systems and users interact. We have evaluated whether and how active and passive recommender systems implement three Human-Computer Interaction indicators: confidence, explanations and feedback.

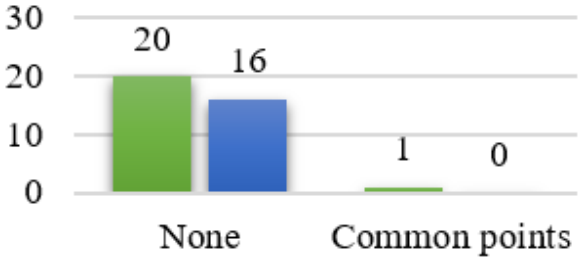
For confidence, we have observed that the assistants either provide a confidence score (i.e., a single value) or they do not provide a confidence




(a) Feedback




(b) Confidence




(c) Explanations

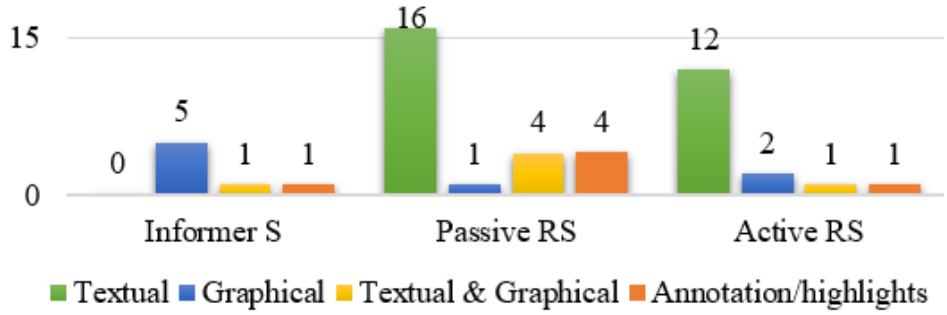
Figure 3.7 – Number of assistants per HCI indicator

indicator at all. Figure 3.7b shows the number of passive and active recommender systems for each group and we can observe how the majority of assistants do not provide any measure of the confidence of their recommendations.

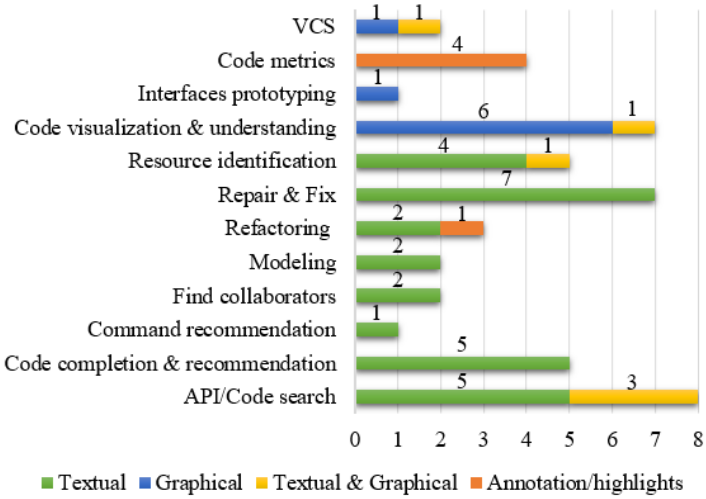
The case is even more accentuated in the case of explanations. Figure 3.7c shows that only 1 assistant out of the 37 (2.7%) provides explanations. This means that, even if some assistants present confidence metrics (e.g., precision), they still act as black-boxes and do not explain the reason behind their suggestions.

We have observed that only three recommender systems [115, 29, 138] include a feedback system and, therefore, let their users provide feedback, as Fig. 3.7a presents. In [29] the recommender system allows users to rate the quality of the suggestions using a rating bar. In [115], the system lets the users reject or modify the suggestions keeping track

of these actions. Finally, in [138], the authors created a recommender that, by using a sensitivity feedback system, lets users adjust the recommendation confidence threshold.




(a) Number of assistants grouped by type




(b) Number of assistants grouped by purpose

Figure 3.8 – Nature of the output provided by the assistants

**Nature of the output provided by the assistants** We have analyzed what the nature of the output that the assistant provide to the users look like and have identified that these outputs are: textual, graphical, textual & graphical, and by means of annotations or highlights. Figure 3.8 presents the nature of the output of the 47 considered assistants grouped by type of assistant and by its purpose. It is worth noticing that there are no informer systems that provide only textual information and that most of them display its results graphically. Unlike informer systems, both passive and active recommender systems, usually display their recommendations textually. These points are supported by the purposes of those assistants, for instance, an assistant that recommends code is likely to show code (text), and an informer assistant that is created for code visualization and understanding is likely to present its results

graphically.

**RQ3: What kind of software technologies are used to embed knowledge in software assistants?**

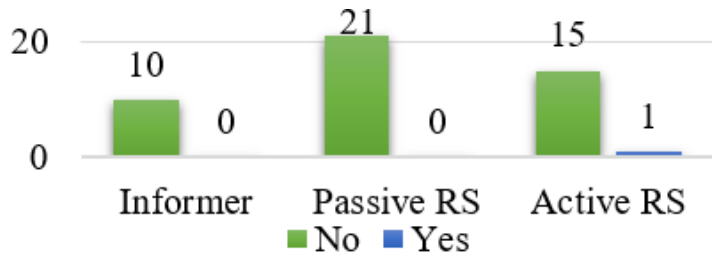



Figure 3.9 – Machine Learning usage

To answer this research question, we have studied whether the assistants embed Machine Learning techniques to simulate *intelligence* and human-like decisions/actions, and what sources of information/knowledge they use.

In Fig. 3.9, we can see that none of the informers nor passive recommender systems use machine learning and that, surprisingly, only one active recommender system uses it. It uses logistic regression to learn how to rank the recommended alternatives before presenting them to the users.

Table 3.8 lists all datasources exploited by the 47 software assistants under study. We have grouped all these datasources in different categories (column 2) and in two groups: local (LOC) and remote (REM), as shown in column 1.

Figure 3.10 presents details about the datasources used aggregated by assistant type. We can see that informer systems do not usually use remote datasources, which is in line with what could be expected as these analyze data at hand to inform about the status of a particular environment. In those cases in which a remote repository is used, these contain information about the working environment (versioning repository, bug tracking platform) and are never general knowledge sources. It is also worth noticing that passive RS use more knowledge sources than active RS.

**RQ4: To what extend are software assistants automated?**

In order to answer this research question, from each primary study, we extracted user-assistant interaction schemes in the form of UML activity diagrams. The full set of the interaction schemes that we extracted is

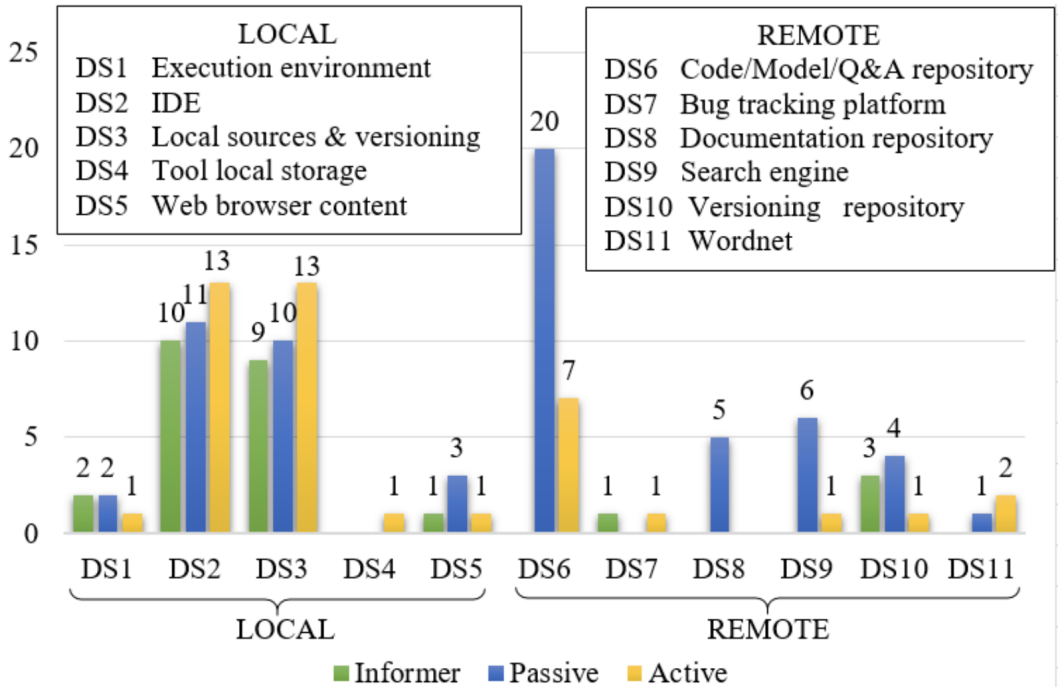



Figure 3.10 – Datasource usage by assistant type

available online[156]. From these schemes, we noticed that systems were either triggerred manually by the user, or automatically by the system: 50% of informer systems and passive recommender systems are system-triggered, while only 23% of active recommender systems are system-triggered.

Automation levels were extracted from these schemes based on the indications provided in [130].

Figure 3.11 presents the extracted automation patterns. We can observe how the analysis is always fully automated—this was a requirement imposed by our characterization of assistants, since without this step, a piece of software would not be considered an assistant but a tool. On the contrary, the decision making process is never automated and the final decision is always made by the user. This means that there is not a single assistant that acts autonomously without "consulting" first—the fact that assistants are designed and created this way could be related to acceptability and trust issues. We would also like to point to the fact that information acquisition is always automated for informer systems, which confirms their role to present relevant information without having the user indicate what to take as input. Figure 3.12 presents the number of assistants that implement these patterns aggregated by assistant type.

| # | Assistant type | Information Analysis | Action Implementation | Information Acquisition | Decision Selection |
|---|---|---|---|---|---|
| 1 | Active RS | Fully automated (10) | Fully automated (10) | Fully automated (10) | Not automated (2-5) |
| 2 | | | | Not automated (2-5) | |
| 3 | Informer | Fully automated (10) | Not supported (1) | Fully automated (10) | Not supported (1) |
| 4 | Passive RS | Fully automated (10) | Not supported (1) | Fully automated (10) | Not automated (2-5) |
| 5 | | | | Not automated (2-5) | |

Figure 3.11 – Automation patterns

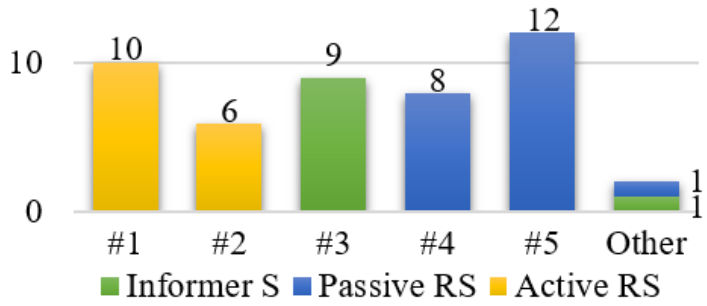



Figure 3.12 – Number of assistants implementing each automation pattern

## 3.4 Limitations and Threats to Validity

This section presents a discussion about the construct validity, the external validity, and the internal validity of the presented results.

### 3.4.1 External validity

The external validity is concerned with whether we can generalize the results outside the scope of our study. We identify three potential threats to external validity for our results.

Firstly, we have restricted the scope of our study to articles published from 2010 onwards and covering the following subjects [27]: Software Design Tools, Software Construction Tool and Software Maintenance Tools, and the extended subject of Socio-Cultural systems according to [140]. This naturally limits the generality of our conclusions to other phases of the software engineering process.

Another threat to external validity relates to our primary studies. This review presents 47 software assistants identified with the protocol in Section 3.2. First, it is possible that some existing assistants were

missed, for instance, because they are not published in research papers or did not match our queries. To mitigate this concern as much as possible, our search contained a large set of keywords with wildcards to maximize matches. Second, our search was limited to the number of peer-reviewed venues that guarantee the quality of the resulting articles, but we may have missed some papers that are not published in these venues or indexed in dblp.org. To mitigate these concerns, a snowballing phase composed by two iterations was conducted.

Finally, the last threat to external validity is linked to the nature of the considered systems. This work studies implemented prototypes or mature systems that can actually be used and tested. Therefore, it excludes articles presenting new algorithms and techniques which are not part of a usable software. This ensures a focus on systems that are really and directly ready to use by software engineers, but may have missed promising potential assistants that are not mature as of today.

We consider that this study has been validated through its systematic protocol and by the different reviews and discussions internally conducted by the authors. This work also provides all the required information for the mapping study to be replicated, which may reduce some external validity concerns.

### 3.4.2 Construct validity

Construct validity refers to using the right tools and tests to get the right measures.

In the context of this systematic mapping study, the most critical point was the extraction of information from the articles. To the extent of our knowledge, the protocols and tools that we used and explained in Section 3.2 are the most appropriate.

Most of the extracted data represents nominal variables, with no intrinsic ordering. This is the case, for instance, of the papers' metadata, supported languages, or even datasources, which cannot be framed within a specific scale. For the identification of other variables such as the characteristics of the recommender systems (e.g., the nature of the output, the explanation system, the confidence indicator, and the feedback system), we relied on the classification of design decisions proposed by Mens and Lozano [112], and for the automation level we used the scales in [185] to match the needs of the analysis.

### 3.4.3 Internal validity

Internal validity is defined as the extent to which the observed results are reliable and lack methodological errors.

The search process relies on an algorithm which performs automated queries, and hence prevents any human error. The exclusion process is the most sensitive part of the exploration protocol. It was conducted manually, and resulted in the rejection of 3,513 papers. In order to limit the subjectivity of this treatment, the exclusion was conducted by only one of the authors, following a rigorous protocol defined in section 3.2. Additionally, before performing the exclusion, an Inter-Rater Reliability score was computed with another author to verify compliance with the criteria.

The data extraction was also performed manually, and consisted in reading all the considered papers and filling in a table with the criteria defined in the protocol. This could lead to errors due to human misinterpretation of the content of the article, missing information or grey areas of the article. However, the content of the articles was sufficient to obtain all the required information. In addition, articles whose information was considered ambiguous were checked at least twice, at different times, in order to minimize errors due to fatigue.

## 3.5 Discussion, open lines of work and challenges

In this section, we summarize the main findings for each research question, and identify eventual research challenges that could drive innovation in the field of software assistants for software design, construction and maintenance.

### 3.5.1 R.Q. 1: What are the tasks that the assistants help their users achieve, in which environments do they operate and which languages do they support?

We found that most software assistants focus on helping software engineers with code-related tasks. Assistance with other tasks such as modeling, finding collaborators or learning IDE commands remains anecdotal. This shows that the literature focuses more on supporting software construction tasks rather than software design or maintenance tasks. Hence, we identify assistance systems for software design as

an open line for research, to support a broader variety of tasks and participants within the software development life cycle.

68% (32 over 47) of the software assistants support Java, while most of the other identified programming languages are only supported by one assistant over 47. Therefore, 40 over 47 (85%) software assistants are mainly integrated in Eclipse IDE, or are not part of any specific development environment (respectively, 49% and 36%). One reason for this distribution could be the large availability of resources for Eclipse plugin development, Eclipse mainly being used as a Java IDE. This technical motivation, however, might take the research away from the actual practices and needs of software engineers. Stack Overflow surveys from 2018 to 2020 show that Java is only the 5th most commonly used programming language, behind JavaScript, HTML/CSS or Python. Eclipse is the 8th most commonly used IDE[4], behind Visual Studio or IntelliJ. Although these historical technologies remain important, new research work must lead innovation on these popular technologies and their inherent technical and human issues. Thus, another research challenge consists in bringing existing or new assistance mechanisms to these increasingly popular environments and languages.

It is worth noting that most of the software assistants created as standalone application or websites have an alternative integrated to an IDE for the same task (see Table 3.5). The papers presenting these systems do not provide an explanation for this design decision, which might also be justified by the ease of development of a website or standalone application compared to an IDE-embedded solution. However, repeated changes of work environment (e.g., switching from the IDE to the web browser) have been correlated with interruptions in the cognitive process, which in turn are correlated with productivity losses [15, 131]. Such human-centric considerations might also be taken into account, in addition to the quality of the algorithm, in order to create truly efficient systems. The following research question emphasizes this notion, while showing that the existing literature performs poorly on such HCI aspects.

### 3.5.2 R.Q. 2: How do software assistants assist users?

This systematic mapping study identified three major types of assistants which respectively have increasing competences: Informer Systems, Passive Recommender Systems, and Active Recommender Systems. Some tasks call for one specific assistant type, such as providing code

[4] 2020 survey do not provide information about IDE.

metrics, which only requires Informer Systems. Other tasks require recommender systems, which could in turn be *passive* or *active*. Unfortunately, papers featuring Passive Recommender Systems did not justify the reason why they do not implement the related action mechanism, which stands for *Active* in Active Recommender Systems. This calls for further work in the direction of exploring how passive recommender systems can be turned into active recommender systems (if applicable) by coming up with a set of actions to perform and execution capabilities.

The acceptability and usability of information systems strongly relies on the relation of trust between one user and the system [14]. This confidence is obtained, among other methods, by allowing the user to understand how the system works and to control it to influence its behavior [41]. Our results about HCI indicators clearly state that the user interfaces of recommender systems hardly support these human aspects. When analyzing the 37 identified recommender systems, 27% display a confidence indicator for their results, 8% allow users to provide feedback on their recommendations, and less than 3% provide an explanation about their recommendations. Studying these HCI aspects, such as how information is presented, how transparency is achieved, and how users control the system, is one major research challenge for software assistants, in order to make algorithms part of a whole *user experience* [69, 73].

### 3.5.3 R.Q. 3: What kind of software technologies are used to embed intelligence in software assistants?

Software assistants must hold a minimal degree of human-like intelligence to be able to perform data analysis and produce new information. This is achieved by implementing hard coded rules and algorithms, which exploit popular technologies and libraries to produce results, with or without the need of external data or knowledge. Our analysis shows that only one software assistant out of the 47 fully implemented systems we studied exploits machine learning techniques. This enables this particular system to change its behavior by learning from examples. While the identified papers do not motivate their choice not to embed machine learning, artificial intelligence or even cognification mechanisms, one open line of work would be to investigate the use of such techniques to support software assistants adaptability regarding user preferences and profiles.

In order to perform analysis and provide recommendations, software assistants exploit data which they obtain locally and/or remotely. Local

information represent the artifacts edited by software engineers (e.g., the source code), the state and history of their IDE, and eventually other client-side information. Remote information describes the data created by co-workers, third-parties or the community (e.g., source code, the content of company databases or model repositories) as well as answers to questions on social community websites (e.g., Stack Overflow), or available documentation (e.g., system requirements). Our results first show that, as expected, Informer Systems mainly use data from local sources (almost 85% of their datasources usage). However, it appears that Passive Recommender Systems tend to exploit much more data from remote data sources than Active Recommender Systems (58% versus 29%). Based on the nature of this data, one may think that current Passive Recommender Systems are likely to be more accurate and "intelligent" than the existing Active Recommender Systems since they are exploiting external knowledge more heavily. One potential reason to justify this result could be the effort required to build that extra step that makes a Passive Recommender system into an Active Recommender System. This leads to another identified research challenge which is to investigate how to facilitate the integration and exploitation of community data, information and knowledge into the analysis algorithms of software assistants with the goal to empower them.

Remote datasources rely on the availability of artifacts and knowledge created by the community. Aggregating these elements creates a global knowledge database which references many general concepts, sometimes called *background knowledge*. This background knowledge can help software engineers deal with common issues, related to general concepts. Building curated and reliable code and data repositories represents another open challenge for the research in software assistants.

### 3.5.4 R.Q. 4: To what extend are software assistants automated?

We have considered the automation of the software assistants in two distinct phases: (i) the activation of the system, i.e., the trigger mechanism and (ii) the behavior of the system once started. Our results show that software assistants are either triggered manually by the user (User Event trigger) or automatically by the system (System Event trigger). Only 23% of Active Recommender Systems are triggered automatically, compared to 50% of Informer Systems and Passive Recommender Systems. This may be due to the fact that Active Recommender Systems can create, modify or delete content, thus, presenting more risks to be automated,

as it is the case for other AI-empowered systems [55].

The extracted automation patterns for each system revealed that software assistants follow one or two interaction patterns according to their type. To date, no software assistant is completely automated from information acquisition to action implementation. This implies that software engineers are always involved in interacting with the system.

Information analysis is fully automated for all software assistants, which aligns with our definition of software assistant. When supported (only for Active Recommender Systems), action implementation is fully automated. Information acquisition is fully automated for Informer Systems, while it could either be poorly automated or fully automated for recommender systems. It is worth noting that only systems with a fully automated information acquisition step are triggered automatically, while the others require the users to trigger them manually. When supported (for Recommender Systems), decision selection remains the least automated part of the process, being poorly automated. The poor automation of these steps might be the consequence of technical limitations (such as user intent acquisition, or knowledge access) or acceptability issues (users might want to keep control over the system). We identify these concerns as another research challenge, which consists in exploring new automation configurations for software assistants, while studying its impact on acceptability.

We encourage researchers to think outside the box and assess new interaction patterns for software assistants. We believe that this classification might be useful to tool vendors who want to include assistance systems into their software solution. Exploiting existing interaction patterns is in line with Jakob's law [123], which states that users prefer a new system to work the same way as all the other systems they already know.

More details about the interaction patterns of the 47 software assistant under study are available online [156].

## 3.6 Conclusions

Since 2010, many types of assistants have become mainstream. For instance, voice assistants like Amazon Alexa and Apple's Siri and the exponential adoption of chatbots in e-commerce. In this chapter, we have studied whether this same trend is happening in software engineering, i.e., whether assistants are becoming a mainstream tool to speed up software development projects. And if so, what the most popular types of assistants are and how they work.

We have observed that the number of research articles introducing fully-finished and ready-to-use software assistants for software design, construction and maintenance tends to decrease. Furthermore, the assistants featured in our set of primary studies are not very well aligned with the software engineers' workflows and preferred programming languages and development environments. We also identified that these works do not take advantage from the recent progress of research in the fields of ML or HCI for information systems. This potentially reduces the effectiveness of the created systems, while limiting their usability and thus their acceptability. Both problems are even more important as the number of smart IDEs from industry continues to increase (e.g., IntelliCode[5], Kite[6], Codota[7], TabNine[8]), in contrast to the number of related research papers.

Thus, research in the field of software assistants for software engineering seems to lag behind the practices and techniques available to date. We see this as a strong opportunity to develop a new generation of assistants that embraces some of the new research results (e.g., ML-based recommenders) and adapts them keeping in mind the findings challenges we described above.

At the same time, we think it is important to pay attention to the evaluation of this new breed of assistants. Rigorous research requires the replicability of the published work in order to compare and evaluate new solutions within the same field. This does not seem to be the case yet in this domain. Only 11 articles out of the 47 primary studies have made a dataset available to provide a benchmark for future comparisons. Finally, another important weakness that we have detected is that only 45% of these user-centric systems conducted an evaluation with real users.

## Contribution from our research approach

This chapter aimed to investigate the existing literature about software assistants for software engineering, to identify their common design characteristics and better understand the literature landscape. At this point of our research approach, two grey areas still challenge the validity of software assistants as a solution to modeling problems. Clarifying these two points would drive our understanding of the context of use and

[5] https://visualstudio.microsoft.com/services/intellicode/

[6] https://kite.com/

[7] https://www.codota.com/

[8] https://tabnine.com/blog/deep

the user requirements, the first two steps of our user-centered research approach.

First of all, the limited presence of modeling assistants in the literature (2 out of 47) may indicate the low interest of the community or of the practitioners for this kind of systems. The low interest of practitioners would invalidate the chance of software assistants to propose a concrete solution to modeling problems. Hence, to validate our approach, it is necessary to collect the feedback about practitioners to better understand their need for assistance.

In Chapter 2, we showed that software assistants were a viable solution to the modeling problems identified in the literature. However, beyond identifying modeling issues, research papers did not propose concrete ways to build software assistants for software modeling. The results of our systematic mapping study did not indicate precise design patterns to follow for the design of modeling assistants. Thus, in order to create modeling assistants tailored to users, it is necessary to investigate how they would like to be assisted.

To address these two aspects, the next chapter presents a study of the software modeling population based on interviews. It describes our protocol and results for the interviews of 16 modeling expert practitioners.

Table 3.5 – Assistant purposes and specific tasks

| Purpose | Specific task | Environment | Lang./Syntax | # | Primary Studies |
|---|---|---|---|---|---|
| API/Code search | Enhances default code completion system | Eclipse IDE | Java | 1 | [8] |
| | Recommends code blocks from text query | Flex Builder IDE | Flex Builder* | 1 | [29] |
| | | Web browser | Java | 4 | [189, 38, 195, 192] |
| | | | jQuery | 1 | [196] |
| | Recommends new features with code | Web browser | Java | 1 | [110] |
| Code completion & recommend. | Enhances default code completion system | Eclipse IDE | Java | 4 | [197, 121, 155, 49] |
| | Recommends code blocks from code analysis | Eclipse IDE | Java | 1 | [177] |
| Code metrics | Augments code with indicators | Brackets IDE | Javascript | 1 | [98] |
| | | Eclipse IDE | Java | 2 | [19, 42] |
| | | Impromptu IDE | Impromptu* | 1 | [176] |
| Code visualization & understanding | Displays code call graphs | Eclipse IDE | Java | 1 | [94] |
| | Displays library dependencies of project | Web browser | Java | 1 | [17] |
| | Finds code responsible for graphical behaviour | Standalone App. | Java | 1 | [87] |
| | Folds less informative code regions | Web browser | Java | 1 | [59] |
| | Proposes new code navigation system | XCode IDE | XCode* | 1 | [86] |
| | Represents code as UML models | Web browser | Java | 1 | [50] |
| | Suggests code locations to explore | Eclipse IDE | Java | 1 | [96] |
| Command recommendation | Recommends tool commands to use | Eclipse IDE | N.A. | 1 | [187] |
| Find collaborators | Suggests potential collaborators | Standalone App. | All | 1 | [175] |
| | | Web browser | Java | 1 | [179] |
| Interface prototyping | Suggests examples for interface prototyping | Web browser | N.A. | 1 | [95] |
| Modeling | Provides a model search engine | Web browser | XMI models | 1 | [103] |
| | Recommends business process model chunks | Standalone App. | BPMN | 1 | [91] |
| Refactoring | Suggests code refactorings | Eclipse IDE | Java | 2 | [115, 180] |
| | | IntelliJ IDEA IDE | Java | 1 | [168] |
| Repair & Fix | Recommends error-related resources | Visual Studio IDE | Visual Studio* | 1 | [89] |
| | Recommends model fixes | Eclipse IDE | UML | 1 | [125] |
| | Recommends Q&A posts | Eclipse IDE | Java | 2 | [43, 143] |
| | Suggests code fixes | Eclipse IDE | Java | 2 | [153, 118] |
| | | Web browser | C++ | 1 | [70] |
| | | | Java | 1 | [70] |
| Resource identification | Recommends code-related resources | Eclipse IDE | Java | 1 | [159] |
| | Recommends error-related resources | IntelliJ IDEA IDE | Java | 1 | [139] |
| | | | JSON | 1 | [139] |
| | | | XML | 1 | [139] |
| | Recommends libraries to add for project | Eclipse IDE | Java | 1 | [122] |
| | Recommends Q&A posts | Eclipse IDE | Eclipse* | 1 | [10] |
| | | | Java | 1 | [138] |
| Version Control System (VCS) | Displays VCS potential conflicts | Eclipse IDE | All | 1 | [71] |
| | | Standalone App. | All | 1 | [31] |

Table 3.6 – Software Assistant types

| Type | Analyze & Display | Help Deciding | Perform Actions |
|---|---|---|---|
| Informer S | Yes | No | No |
| Passive RS | Yes | Yes | No |
| Active RS | Yes | Yes | Yes |

Table 3.7 – Assistant types for specific tasks

| Purpose | Specific task | Type | # | Primary studies |
|---|---|---|---|---|
| API/Code search | Enhances default code completion system | Passive RS | 1 | [8] |
| | Recommends code blocks from text query | Active RS | 1 | [29] |
| | Recommends code blocks from text query | Passive RS | 5 | [196, 192, 38, 189, 195] |
| | Recommends new features with code | Passive RS | 1 | [110] |
| Code completion & recommendation | Enhances default code completion system | Active RS | 4 | [155, 197, 49, 121] |
| | Recommends code blocks from code analysis | Passive RS | 1 | [177] |
| Code metrics | Augments code with indicators | Informer S | 4 | [42, 19, 98, 176] |
| Code visualization & understanding | Displays code call graphs | Passive RS | 1 | [94] |
| | Displays library dependencies of project | Informer S | 1 | [17] |
| | Finds code responsible for graphical behaviour | Passive RS | 1 | [87] |
| | Folds less informative code regions | Informer S | 1 | [59] |
| | Proposes new code navigation system | Informer S | 1 | [86] |
| | Represents code as UML models | Informer S | 1 | [50] |
| | Suggests code locations to explore | Active RS | 1 | [96] |
| Command recommen. | Recommends tool commands to use | Passive RS | 1 | [187] |
| Find collaborators | Suggests potential collaborators | Passive RS | 2 | [175, 179] |
| Interfaces prototyping | Suggests examples for interface prototyping | Active RS | 1 | [95] |
| Modeling | Provides a model search engine | Passive RS | 1 | [103] |
| | Recommends business process model chunks | Active RS | 1 | [91] |
| Refactoring | Suggests code refactorings | Active RS | 3 | [180, 115, 168] |
| Repair & Fix | Recommends error-related resources | Passive RS | 1 | [89] |
| | Recommends model fixes | Active RS | 1 | [125] |
| | Recommends Q&A posts | Passive RS | 2 | [43, 143] |
| | Suggests code fixes | Active RS | 3 | [118, 153, 70] |
| Useful resources identification | Recommends code-related resources | Passive RS | 1 | [159] |
| | Recommends error-related resources | Passive RS | 1 | [139] |
| | Recommends libraries to add for project | Active RS | 1 | [122] |
| | Recommends Q&A posts | Passive RS | 2 | [10, 138] |
| VCS | Displays VCS potential conflicts | Informer S | 2 | [31, 71] |

Table 3.8 – Datasources description

| Type | Category | Datasource | # | Primary Studies |
|---|---|---|---|---|
| LOC | Execution environment | JVM Access | 2 | [87] [19] |
| | | Access to runtime environment | 1 | [176] |
| | | ByteCode Access | 1 | [87] |
| | | Java Binaries | 1 | [49] |
| | IDE | IDE Access | 32 | [42] [118] [168] [96] [197] [155] [91] [87] [122] [180] [139] [8] [115] [153] [177] [98] [89] [121] [187] [19] [138] [31] [49] [50] [94] [176] [70] [10] [17] [86] [71] [159] |
| | | IDE Cache | 1 | [42] |
| | | Tool commands registry | 1 | [187] |
| | Local sources and versionning | Sources + Versionning local | 32 | [42] [118] [168] [96] [197] [155] [91] [87] [122] [180] [139] [8] [115] [153] [177] [98] [89] [121] [187] [19] [138] [31] [49] [50] [94] [176] [70] [10] [17] [86] [71] [159] |
| | Assistant local storage | Collection Database | 1 | [96] |
| | Web browser content | Access to webbrowser request and content | 3 | [139] [98] [159] |
| | | Access to internet web searches | 1 | [189] |
| | | Access to Web Browser Editor | 1 | [95] |
| REM | Bug tracking platform | Bug Report access & failing test | 1 | [153] |
| | | Runtime Execution Traces | 1 | [42] |
| | Code/Model/Q&A repository | Stack Overflow database | 7 | [43] [196] [38] [89] [138] [143] [10] |
| | | Source Code Corpus | 4 | [155] [122] [177] [121] |
| | | API Libraries | 3 | [192] [189] [38] |
| | | Model Corpus | 2 | [91] [103] |
| | | Android Sources | 1 | [195] |
| | | API Sentence Model | 1 | [155] |
| | | Bytes.com access | 1 | [89] |
| | | Codeguru.com access | 1 | [89] |
| | | Daniweb.com access | 1 | [89] |
| | | DevShed access | 1 | [89] |
| | | Feature Request List | 1 | [192] |
| | | Fix Library | 1 | [70] |
| | | Github Rest API | 1 | [179] |
| | | Source Forge code repository | 1 | [175] |
| | | Web pages corpus | 1 | [95] |
| | Documentation repository | Documentation Access | 3 | [192] [189] [38] |
| | | Android Doc | 1 | [195] |
| | | Software Documentation | 1 | [110] |
| | Search engine | Google | 3 | [29] [138] [143] |
| | | Bling access | 2 | [138] [143] |
| | | Blekko access | 1 | [138] |
| | | Yahoo access | 1 | [143] |
| | Versionning repository | Specific Source Code Repository Access | 7 | [192] [125] [110] [8] [31] [59] [71] |
| | | Closed feature request repo | 1 | [192] |
| | Wordnet | Wordnet | 3 | [195] [91] [49] |

Chapter 4

# The need for assistance in software modeling practice

*While software assistants theoretically appear as potential solutions to tackle modeling issues, this remains to be proved empirically. For modeling assistants to be tailored to users' needs, we first need to collect the modeling engineers' needs and expectations about modeling assistance. Addressing these two issues is part of our user-centred research approach, that first consists in (i) understanding and specifying the context of use of the system and (ii) specifying the user requirements.*

*This chapter presents the results of a series of interviews conducted with 16 software modeling experts as a first qualitative research effort investigating the need for assistance in modeling. It allowed us to gather user requirements about the design of software modeling assistants, but also enabled us to strengthen our understanding of modeling assistance. From the analysis of our conversations, we formulate a set of 12 hypotheses, which pave the way for the design of user-tailored modeling assistance systems and call for further evaluation. This work led to the writing of a journal article that has been submitted, and which is currently under review.*

## 4.1 Study design

In this chapter, we investigate the position of software modeling experts about modeling assistance and software modeling assistance. To do so, we planned multiple interview sessions with experts from various

companies and business domains. In this section, we introduce our research questions, and present the research method that we followed for this study. We conducted 16 structured interviews with modeling experts, exploiting a questionnaire of 49 items to address our 5 research questions.

### 4.1.1 Research questions

Our work aims to answer the five following research questions. Questions 1 to 4 were split in several subquestions to refine their goal, and to present the results in a clear and organized manner.

- *R.Q. 1: Does the profile of the participants meet the profile of software modellers as presented in the literature?*
  - *R.Q. 1.1: How often do they model?*
  - *R.Q. 1.2: What are the tools they use?*
  - *R.Q. 1.3: What kind of diagrams do they realize?*
  - *R.Q. 1.4: Why do they model?*
  - *R.Q. 1.5: What are the most challenging aspects of modeling for them?*
- *R.Q. 2: What do software modeling experts expect from the ideal software modeling assistant?*
  - *R.Q. 2.1: Do they need help when modeling?*
  - *R.Q. 2.2: How could the ideal software modeling assistant help them?*
  - *R.Q. 2.3: What makes a good software modeling assistant?*
- *R.Q. 3: How would software modeling experts like to interact with the ideal software modeling assistant?*
  - *R.Q. 3.1: How should the system be triggered?*
  - *R.Q. 3.2: Should the system indicate how confident it is?*
  - *R.Q. 3.3: Should the system explain its suggestions?*
  - *R.Q. 3.4: Should the system provide a feedback mechanism?*
  - *R.Q. 3.5: Could the system display wrong suggestions?*
- *R.Q. 4: How could a software modeling assistant help experts' colleagues?*

  - *R.Q. 4.1: How are participants solicited by their colleagues to help them?*
  - *R.Q. 4.2: How do participants feel about their colleagues' trust in their answers?*
  - *R.Q. 4.3: Could a software modeling assistant act like participants to help colleagues?*
- *R.Q. 5: What limitations could prevent the development of the ideal software modeling assistant?*

### 4.1.2 Research method

We performed a qualitative research study based on structured interviews. We report on our research approach by describing its (i) design and planning, (ii) data collection, and (iii) data coding.

#### Study design and planning

We conducted a semi-structured interview study as a preliminary means to identify opportunities and challenges to assist modeling practitioners. Interviews allowed us to achieve this qualitative research effort while collecting more data from participants than questionnaires. To this end, we built the guide of the interview exploiting techniques from design thinking [92] and created 50 mostly open-ended questions (see Table 4.8) to stimulate participants' thinking and discussion around our four research questions. We encouraged participants to provide the maximum amount of details for all their answers before starting the interview session.

The estimated time of the session was 90 minutes to let participants talk as much as possible. Before scheduling the interview session, participants had to fill a non-disclosure agreement ensuring the confidentiality of the collected content as well as their anonymity. Due to health-related restrictions in place at the time of the study design, the interviews were designed to be conducted via video and audio conferencing. The sessions were audio-recorded for them to be transcribed and analyzed. Two test sessions were conducted with two colleagues who have industry experience to test the technical set-up and the questionnaire. The wording of some questions was modified to facilitate their understanding.

Participants were gathered according to their profile, which should match the following criteria:

- At least a 5 years experience in software modeling,

- Performs software modeling with UML regularly and eventually with other modeling languages,
- Software architect, functional manager, or any other position related to the functional, process, or business aspects of software.

The call to participate in our experiment was first sent by email on a mailing list of corporate human resources contacts internal to the university. We also conducted searches on LinkedIn[1] in order to contact as many practitioners as possible who fit the profile we were looking for.

### Data collection

Invitations to join our study were sent during the month of February 2021. We sent an invitation email through an industry mailing list of the University of Lille and directly contacted 44 LinkedIn profiles. At the end of the invitation process, 16 practitioners matched the profile and signed the non-disclosure agreement. As the participants were all native French speakers, the questions and answers were given in French. The interviews were conducted in March 2021 in a virtual setting using videoconferencing tools such as Zoom[2], Microsoft Teams[3], or Google Meet[4], depending on the participant's company agreements.

The sessions of questions took 83 minutes on average and were followed by a debriefing time of 13 minutes on average. This debriefing moment was dedicated to answering all questions of the practitioners about the research we are conducting and collecting their informal feedback about our questions. A single author, Maxime Savary-Leblanc, conducted the interviews to ensure the consistency of the protocol. In total, 25 hours and 42 minutes of discussion were recorded, from which 22 hours and 13 minutes –dedicated to the questions– had been exploited in this study. To allow a detailed analysis of the responses, the recordings were transcribed into text by a specialized company.

### Data analysis

We analyzed the transcripts and used NVivo[5], a text-coding software enabling fine-grained coding for qualitative analysis. The coding phase

[1] https://www.linkedin.com/
[2] https://zoom.us
[3] https://www.microsoft.com/fr-fr/microsoft-teams/
[4] https://meet.google.com
[5] https://ritme.com/software/nvivo/

was performed manually since searching for keywords could be insufficient, due to practitioners referring to a same idea in different ways. For each interview question, codes were created to comprehensively represent the ideas and opinions expressed by the participants. To conduct the qualitative analysis of the data, we coded as much text as possible related to the questions. The uncoded parts of the text represent elements of communication management or elements that are not related to the questions. This is the case for the presentation of the participant's professional background at the beginning of the session.

The questions asked in the sessions were intended to stimulate the participants' thinking and imagination. Some participants needed more time to think about their answers, others thought orally. Thus, the coding allowed us to analyze the presence or absence of ideas and arguments in the participants' speech, but does not allow us to draw conclusions about the strength of an idea based on the number of times it appears in the speech. In this chapter, the strength of an argument or idea is represented by the number of participants referring to it. However, since the analysis is qualitative, this process is only used to order the ideas or identify possible trends, and should not be considered as providing meaningful statistics.

## 4.2 Results

This section presents the results of the data coding and analysis. We first introduce the demographics about participants, and report on the results following the research questions of Section 4.1.1.

### 4.2.1 Demographics

Questions A1 to A4 of the questionnaire allowed us to collect general information on the participants. This information is presented in Table 4.1 and allows us to draw up a general profile of the respondents. The participants in the study have a median modeling experience of between 10 and 15 years, and represent a cumulative experience of between 200 and 270 years. The distribution between small and large companies appears to be relatively even, and the median is located at companies with 50 to 100 employees. Note that this value represents the company where the participant is an employee. Thus, freelance workers are counted as companies with less than 10 employees but may have much larger companies as clients. This is the case for profiles P3, P8 and P9, who work for clients with over 100 employees. Despite our efforts in

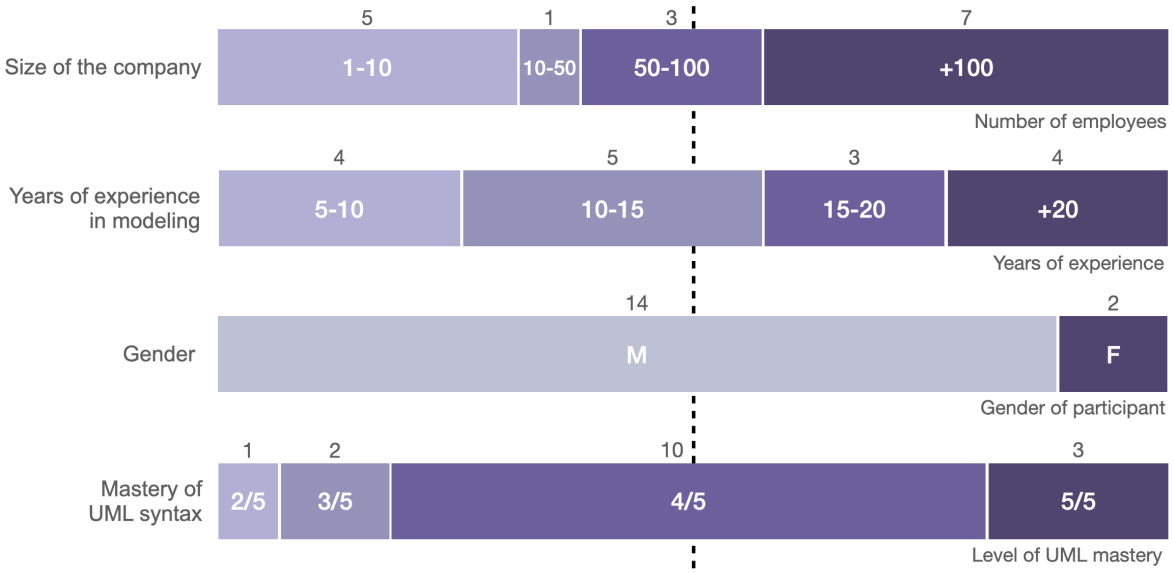
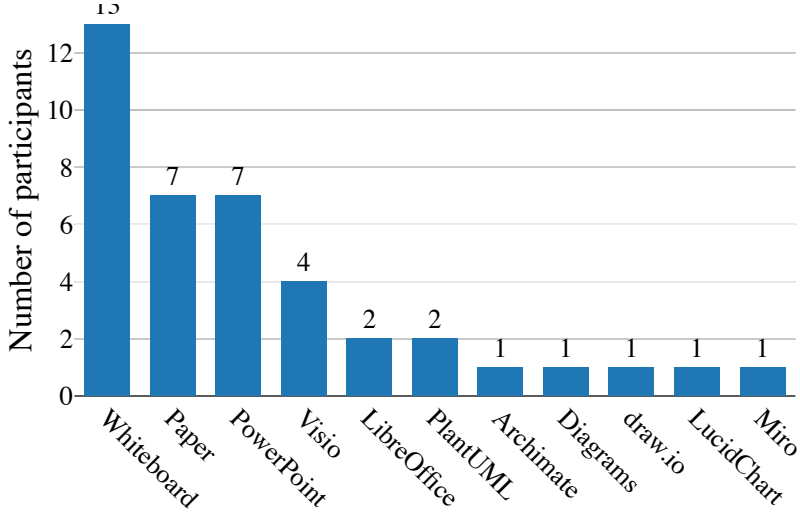



Figure 4.1 – Overview of the panel of participants

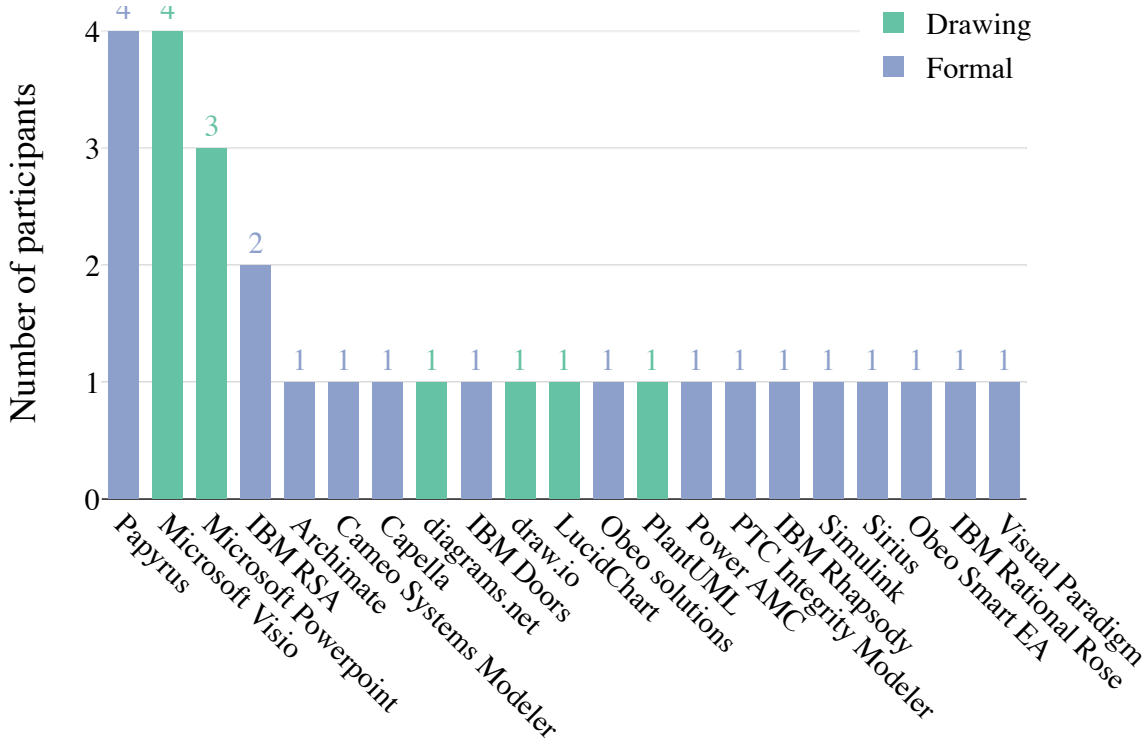
searching for candidates, the population represented is predominantly male (14 out of 16). 13 out of 16 participants rate their UML proficiency at least 4 out of 5 on a 5-point Likert scale, which is in line with the selection criteria. The remaining 3 participants felt that they did not meet the standard strictly enough to give themselves a higher score. As the participants evaluated their communication skills themselves, they might have overestimating or underestimating their skills, which influences the presented results. However, this enables us to assume that all participants have a sufficient command of UML to answer our questions. Our panel of participants covers many domains in industrial production and service delivery. The domain of finance, insurance and pensions is particularly represented as it covers many entities, such as banks, supplementary pension organizations, or governmental public actors.

### 4.2.2 R.Q. 1: Does the profile of the participants meet the profile of software modellers as presented in the literature?

In this section, we report on questions A6 to B4 (see Table 4.8), which investigated the general modeling practice of the participants, in order to better understand their profile. We present results about how often they model, and with what tools. We also introduce results about what kind of diagrams they realize, what they are aimed for, and report the aspects of modeling that they find the most challenging.




(a) Informal modeling media




(b) Formal modeling tools

Figure 4.2 – Formal and informal modeling media

| Id | Gender | Age | Company size | Domain | Position | Modeling experience | UML level |
|---|---|---|---|---|---|---|---|
| P1 | F | 33 | 50-100 | Automotive, aerospace, energy or construction | System Modeling Engineer | 5-10 | 5 |
| P2 | M | 55 | 50-100 | Automotive, aerospace, energy or construction | Head of R&D | +20 | 4 |
| P3 | M | 46 | -10 | Construction materials | Entreprise Architect | +20 | 5 |
| P4 | M | 46 | +100 | Banking, insurance, pensions | Chief Architect | 10-15 | 3 |
| P5 | M | 41 | -10 | Telecoms | Chief Technical Officer | 15-20 | 4 |
| P6 | F | 42 | +100 | Banking, insurance, pensions | Information System Architect | 10-15 | 4 |
| P7 | M | 43 | +100 | Banking, insurance, pensions | Project Manager | 5-10 | 4 |
| P8 | M | 44 | -10 | Machine manufacturing | System Architect | 15-20 | 4 |
| P9 | M | 31 | -10 | Banking, insurance, pensions | Project Manager | 5-10 | 4 |
| P10 | M | 56 | +100 | Research | Head of R&D lab | 15-20 | 4 |
| P11 | M | 42 | 50-100 | IT services | Head of Business | +20 | 5 |
| P12 | M | 35 | 10-50 | Automotive, aerospace, energy or construction | Project Manager | 10-15 | 2 |
| P13 | M | 47 | +100 | Banking, insurance, pensions | IS Manager | 10-15 | 4 |
| P14 | M | 42 | +100 | Banking, insurance, pensions | Head of Data Architecture | 5-10 | 3 |
| P15 | M | 51 | +100 | Banking, insurance, pensions | Architecture governance | +20 | 4 |
| P16 | M | 34 | -10 | Automotive, aerospace, energy or construction | Modeling technical adviser | 10-15 | 4 |

Table 4.1 – Summary of participant's profiles

### R.Q. 1.1: How often do they model?

Questions A10 and B1 were intended to measure how frequently participants practice informal and formal modeling. Interviewees were asked to choose between *daily*, *multiple times a week*, *multiple times a month*, and *never*. Figure 4.3 presents the frequency results for the 16 practitioners for both informal and formal modeling. We refer to formal modeling as *modeling with dedicated tools, with respect to the UML standard, to produce diagrams intended to be reused and stored*, as opposed to informal modeling, that we define as *informally sketching pieces of diagrams, on any media, that are not intended to be reused over time*. Some participants informed us that their modeling workload fluctuates depending on the weeks and projects they are working on. In this case, we asked them to smooth their response so that it is representative of their work in general. 12 participants our of 16 (75%) practice formal modeling regularly, at least multiple times a week while 10 participants (62%) do so with informal modeling. We note that the 4 practitioners who perform formal modeling monthly or less are all managers in charge of one or several teams of employees. Their relationship to modeling is then made through the collaborators they supervise and their response should not be interpreted as *not using formal modeling in their work*.

### R.Q. 1.2: What are the tools they use?

Questions A11 and B2 enabled us to list the media that participants exploit to create UML models. Both questions differentiate media used to

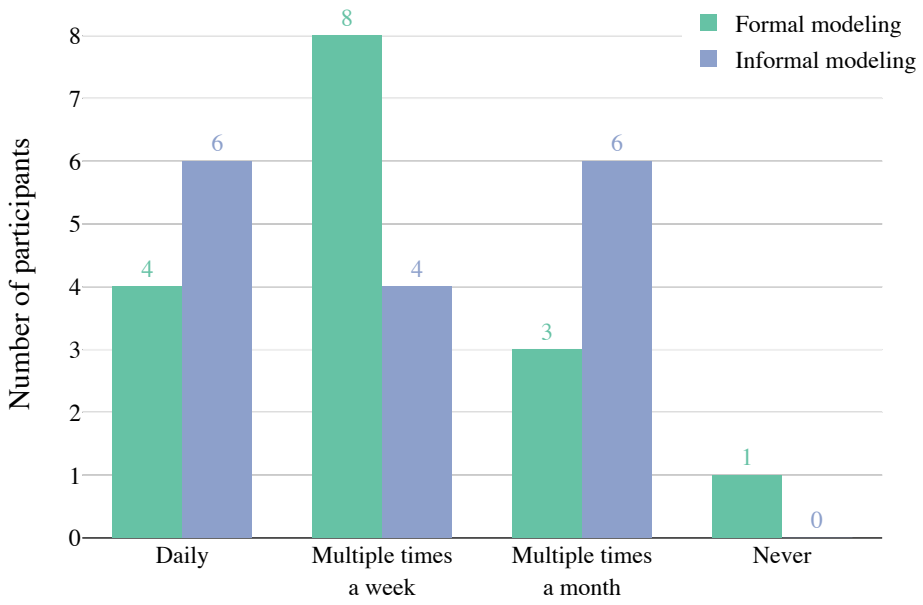



Figure 4.3 – Participants' frequency of both informal and formal modeling

produce informal models from formal models as respectively presented in Figure 4.2a and 4.2b. When producing informal models, participants prefer to use non-digital media such as whiteboard (13 out of 16) or paper (7 out of 16). 11 practitioners use software tools to draw informal diagrams such as Microsoft Powerpoint (7 out of 16), or LibreOffice (2 out of 16). They also use diagram-drawing tools such as Microsoft Visio, diagrams.net or draw.io.

We have identified 21 software tools that allow participants to build formal UML models. Two of the top-3 most used tools (Microsoft Visio and Powerpoint, respectively 4 and 3 participants) are diagram-drawing software that allow for the creation of informal diagrams that do not necessarily respect the UML standards. Participants P5 and P15 switched from formal software to this kind of more simple tools because they identify most of formal modeling software as real mazes, e.g., tools too complex to use for them. We identified 6 drawing tools, and 15 formal modeling tools in interviewees' answers. The participants who chose drawing tools justify this choice by the ease of collaborative work, their light and mobile client, and their affordability. Participants using formal modeling tools cited the imposed choice of the client or company as the main reason, followed by the open-source nature of the software, or its strict compliance with the standard. The high price of many of these tools is also cited as a reason for not being allowed to change, as

companies do not want to pay for several different licenses.

**R.Q. 1.3: What kind of diagrams do they realize?**

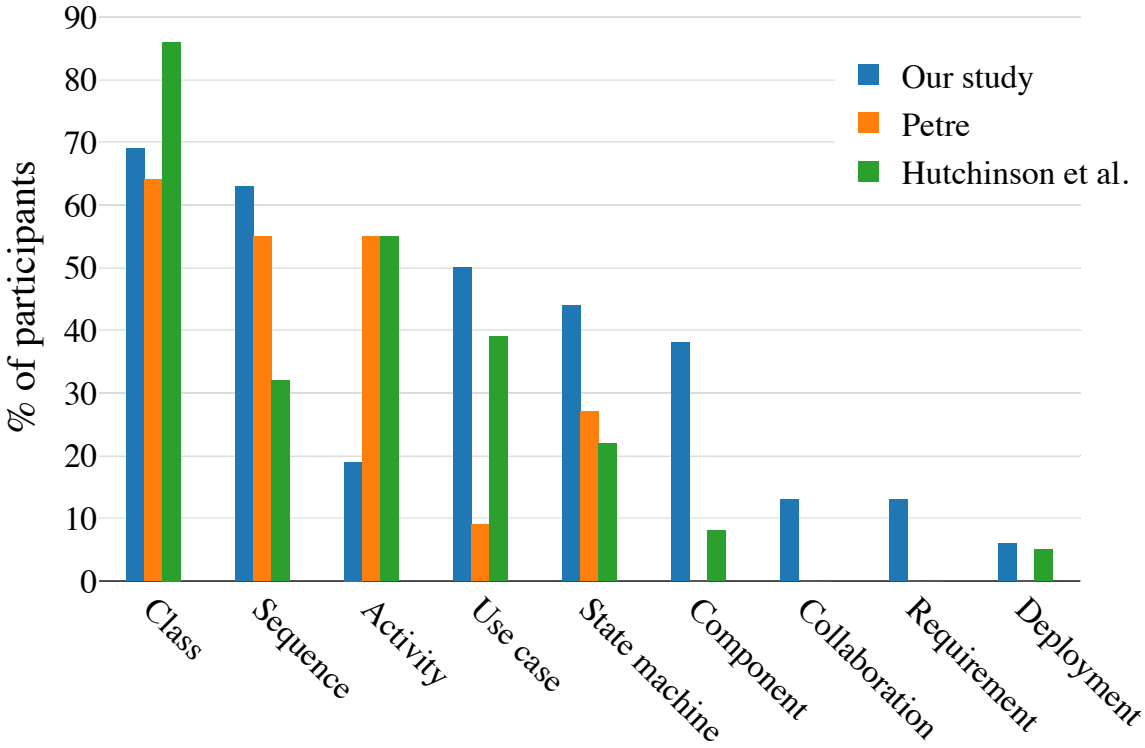



Figure 4.4 – Types of UML diagrams that participants use

Study participants were mainly selected because of their use of UML. Thus, question A7 was aimed to identify the UML diagrams they exploit at work. Figure 4.4 compares the declared usage of diagrams from our study with the results of studies by Petre [136] and Hutchinson et al. [77]. The study of Petre focused on understanding the use of UML in practice while Hutchinson et al. provide an overview of modeling in MDE approaches. The percentage value in the figure represents the amount of practitioners who use each type of UML diagram. In our study, 11 practitioners use class diagrams, 10 use sequence diagrams, 8 use use case diagrams, 7 use state machine diagrams), and 6 use component diagrams. The use of other kinds of UML diagrams remains more sparse according to the considered study. 2 participants (P1 and P2) also declared using SysML IBD and BDD diagrams regularly.

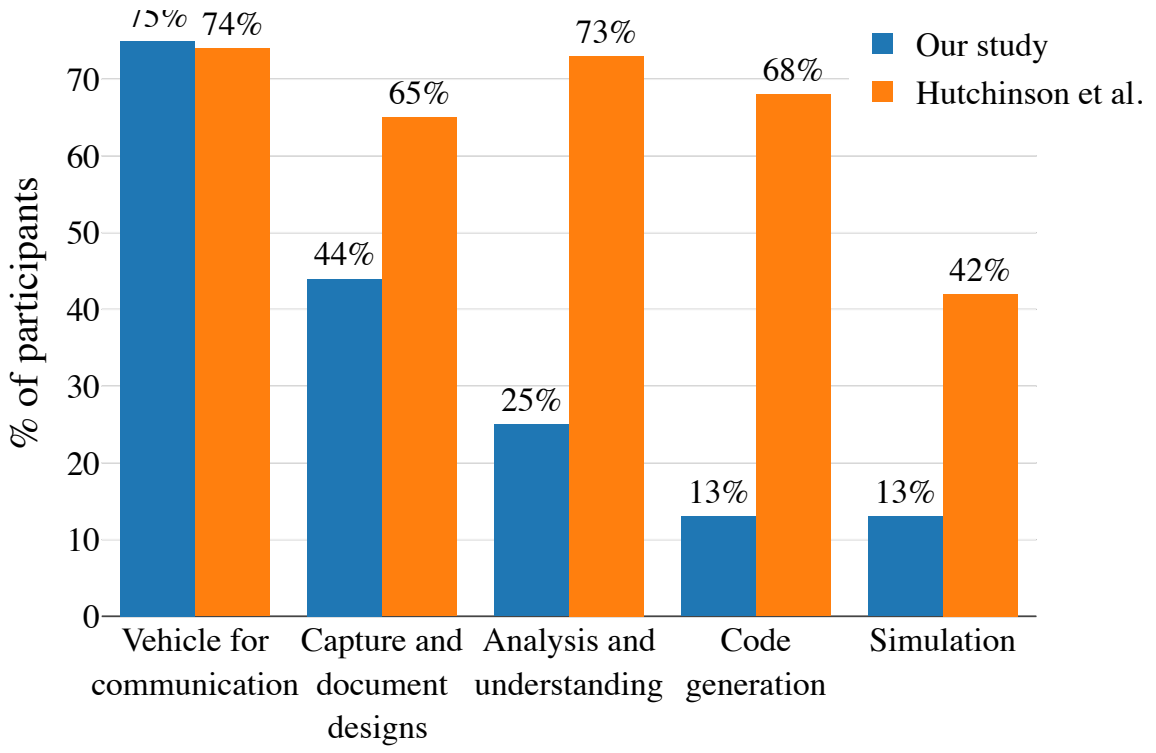



Figure 4.5 – Modeling goals of the participants

**R.Q. 1.4: Why do they model?**

The purpose of question A8 was to understand why participants create models. We coded the results of this question using the classification of Hutchinson et al. [77], established from a previous study. Figure 4.5 introduces why our practitioners use models, and compares our results with the results from their study about MDE.

Our expert modelers mainly use models as a vehicle for communication (12 out of 16). The focus is then mainly on reaching agreement between the different team members as described by participant P5 who uses UML "*to exchange so to communicate with other teams, to share information, to have an unambiguous format, to be understandable by all, to agree on a vision of a requirement and more generally of a business*". As another participant summarized, they "*model to share, to make sure everyone understands the same thing. So it's overall for communication purposes*". 7 practitioners (44%) use models to capture and document software designs while creating requirements or software documentation. The first purpose of creating these documents is to enable the development of the solution, as pointed out by participant P9 who describes that "*the second big use [of models] is to provide these diagrams - when they are*

| Task | # | Domain knowledge | Corporate methodology | Notation | Tool usage | Modeling know-how |
|---|---|---|---|---|---|---|
| Adapt level of detail to the recipient of the model | 7 | | | | | X |
| Understand the domain and the client | 7 | X | | | | |
| Layout diagrams | 5 | | | | X | |
| Tools too complex to use | 5 | | | | X | |
| Modify an element in an existing model | 4 | | | | X | |
| Creating many elements is tedious | 2 | | | | X | |
| Keep the models up to date | 2 | | X | | X | |
| Reuse models | 2 | | X | | X | |
| Choose the right model elements | 1 | | | X | | |
| Know the best way to use the tool | 1 | | | | X | |
| Split and architect the model | 1 | X | | | | X |
| Make a model that is sufficient for understanding | 1 | X | | | | X |
| Copy and paste is complicated | 1 | | | | X | |
| Mastery of UML | 1 | | | X | | |
| Lack of active support for modeling | 1 | X | | | X | |
| Lack of guidance during modeling | 1 | X | | | X | |
| Poor ergonomics of the tool | 1 | | | | X | |
| Use templates | 1 | | | | X | |
| Do not forget elements in the large models | 1 | X | X | | | |
| Respect the UML notation | 1 | | | X | | |
| Adapt to time and budget constraints | 1 | | | | | X |
| Use a drawing tool to model | 1 | | | | X | |
| Total of respondents | | 10 | 4 | 3 | 11 | 8 |

Table 4.2 – Most challenging aspects of modeling according to our participants

*validated by the management - to the developers, to software engineers who are going to rely on them*". In more critical environments, these documents can be created in response to a strong need for validation, as in the case of participant P10, who recalls that for "*space projects, you have to follow very specific protocols, they require an absolutely crazy amount of documentation, of tests, of things, of validation, the SRS, the software requirement, then the software validation, then the tests. There, I'm quoting the important documents, but the document that describes the design, that is, what is the architecture and the design that meets the requirements, the specifications, this document uses a lot of UML.*" Our participants more rarely use models to analyze and understand a system (3 practitioners), to generate code (2 practitioners), or to run simulations (2 practitioners).

### R.Q. 1.5: What are the most challenging aspects of modeling for them?

As a first step to understand how expert practitioners could be assisted when modeling, we identified the tasks that they perceive as difficult. Question B4 was about identifying and listing these challenging tasks. When unclear, we asked participants to refine and detail their answer. We collected 22 tasks that are perceived difficult, and grouped them into five categories according to their nature as shown in Table 4.2.

**Tool usage** relates to all tasks that are directly related to the modeling

tool, such as creating or editing model elements in a diagram.

**Domain knowledge** gathers tasks related to the knowledge and understanding of domain-specific concepts that are necessary to the construction of the model. It covers the general understanding of the field that the practitioners are working for, as well as the mastery of the concepts that should be represented in the diagram they conceive.

**Modeling know-how** refers to the tasks that compose the essential complexity of the modeling activity, which is inevitable, which relates to the very nature of a task, and which cannot be removed. For instance, choosing what to model, or deciding of the model architecture, are tasks that rely on the modeler's experience and are the core of the modeling activity.

**Corporate methodology** covers tasks that are included in company-internal processes. In some companies, operations on models must follow internal guidelines about where to store data, the amount of metadata to include, or naming or modeling conventions. Such guidelines set contraints on the modeling tasks that engineers should perform.

**Notation** relates to tasks requiring the knowledge and the mastery of the UML notation, such as mapping a concept or a rule to the right diagram element.

Participants most complain about the general tool usage (11 out of 16, 69%) and especially about tools being too complex (5 out of 16), with bad ergonomics (1 out of 16). Modeling tools are perceived by participants P5 and P13 as *"real mazes"* with way too many features for their needs. Some specific features are also pointed out such as the layouting (5 out of 16), the editing of existing model elements (4 out of 16), the synchronization of models (2 out of 16), and their reuse (2 out of 16).

Tasks related to domain knowledge also appear difficult for 10 out of 16 participants. During the interviews, practitioners identified two distinct challenges which fall into this category. The first is to gather and understand the customer's need, comprehensively and accurately enough to enable the creation of the model. For example, participant P8 says that the most complicated part is *"understanding the customer. Even though I have 20 years of experience in modeling, it really takes a huge intellectual effort. The problem is not in the modeling, but in understanding what the customer is saying in order to translate what the customer is saying into UML. Yes, for me, understanding the need is the most difficult."* The second challenge is to have general knowledge of the modeling domain. Access to this knowledge can be complex because it can be distributed

between different human or IT resources.

8 participants identify tasks related to modeling know-how as challenges during modeling activity. The biggest challenge is to adapt the level of detail of the model to its recipient. This is a challenge faced by participant P3, who thinks that *"the difficulty lies in knowing how the model will be perceived by the person on the other side. In fact, a UML model, for example, I'm going to send it to a technician who will understand it, but I'm going to send it to a business that won't necessarily understand everything. If I send it to a manager, it's even worse, it depends on the manager."* Participant P15 also shares this point of view by stating that *"the difficulty is finding the right level of abstraction to go and talk to our different stakeholders. What is the right level of abstraction to go and talk to the business? What is the right level of abstraction to discuss with the architects? What is the right level of abstraction to discuss with the people in charge of operations? Everyone has their own habits, their own way of looking at the same subject. The operations guy doesn't care about the business division, it's not his problem. He wants to know physically on which machine the component is running and when the machine breaks down, does he have redundancy? His concern is not at all at the same level of detail as the business. The business doesn't care where the component is instantiated for instance."*

One practitioner also pointed out the difficulty to design and organize the model, while another reported on the complexity of conceiving models that are sufficient for understanding.

### 4.2.3 R.Q. 2: What do software modeling experts expect from the ideal software modeling assistant?

Part B of the questionnaire was focused on practitioners' need for help when modeling, either it is from their colleagues or a digital assistance system. In this section, we report on questions B5 to B11, which investigate how practitioners would welcome such a computerized system, by assessing if they could perceive it as something useful, understanding what they expect from it, and collecting their ideas about what it could do.

#### R.Q. 2.1: Do they need help when modeling?

During the interviews, several practitioners asked for clarification on the concept of help in order to answer question B5. We therefore defined help for the practitioners as the intervention of a third party in order to accelerate their work or to unblock a situation. Thus, any person

answering the participant's questions during the modeling process is considered as assistance.

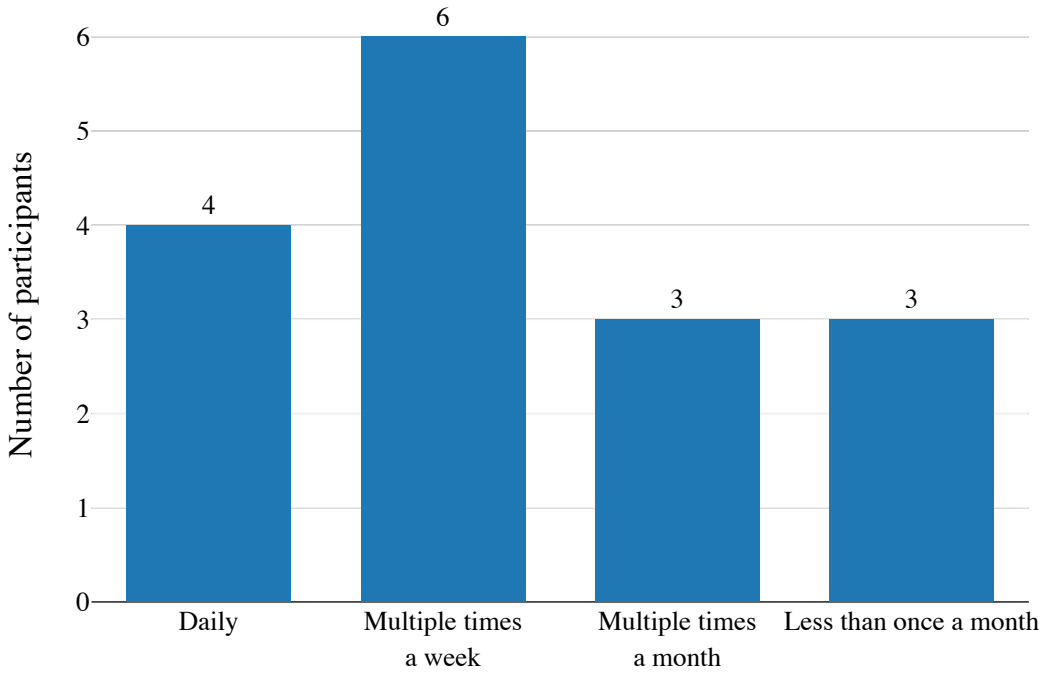



Figure 4.6 – Participants' need for help when modeling

Figure 4.6 shows the frequency of participants' need for assistance. More than half of interviewees (10 out of 16) regularly require help while 3 participants use help less than once a month. This help is mainly related to mastering the business domain as presented in Table 4.3. 9 out of 16 participants required help in understanding the domain and business processes, and 4 participants required help in ensuring the completeness of the model on the same topics. Some participants also required help when searching for useful resources (2 out of 16), learning how to use the modeling tool (2 out of 16), and exploiting the UML syntax the right way (2 out of 16).

### R.Q. 2.2: How could the ideal software modeling assistant help them?

Question B7 introduces the notion of software modeling assistant and was intended to understand practitioners' opinions on the usefulness of such a system. We defined a software modeling assistant (or simply assistant during the interviews) as software capable of helping them, without further precision in order not to bias the answers. The objective was to understand whether a computer system could provide the

| Aspect of the assistance | # of participants |
|---|---|
| Understanding of domain and business processes | 9 |
| Domain completeness of the model | 4 |
| Locate useful information | 2 |
| Usage of modeling tool | 2 |
| Usage of UML | 2 |
| Decision-making with the client | 1 |
| Impact study of changes | 1 |
| Collaborative modeling | 1 |
| Peer review and validation | 1 |

Table 4.3 – Aspects of participants' required modeling assistance

assistance mentioned in Section 4.2.3. 11 out of 16 practitioners (69%) consider that a modeling assistant could help them with these topics today, given the current state of technology and data stored in computer systems. The five other practitioners do not indicate that they would not like to be assisted, but rather mention limitations (detailed in Section 4.2.5) that still prevent such system to exist for them. However, when asked to overlook the current technological limitations, 100% of practitioners envision a system that can help them. Thus, 100% of our participants would welcome a modeling assistant to support their work, provided that the system offers the expected functionality, and works in the expected way.

Questions B10 and B11 asked participants to imagine what assistance features they would like to have when they model. A total of 21 features were mentioned (identified in Table 4.4), covering 4 of the 5 task categories identified in Section 4.2.2. Only the *Modeling know-how* is not represented in the listed features. All participants (16 out of 16) imagined features related to *tool usage*. Such functionalities are mainly intended to reduce the modeling effort or the modeling time. Among the most cited features, 6 practitioners would like the assistant to be able to generate diagrams autonomously. This refers to the ability of the system to create an initial diagram about a business domain, with recommended elements already added, as a template. Participant P13 imagines a system capable of *"automatically generating the diagrams, I don't need to do it again since I have already entered the data, through other diagrams and according to the existing"*. For participants P8 or P15, it is more a question of refining or transforming the model to another representation. For them, *"if the assistant existed, ideally it would allow, from the model that was originally made, to be able to generate almost another model, a little more precise, technical, of implementation"*.

5 participants would like to be provided with model templates when starting modeling. Participants also imagined an assistant that could layout diagrams autonomously (4 out of 16), or that could ensure the consistency of several models or diagrams (4 out of 16). In addition, some practitioners suggested different ways to interact with the modeling tool, such as being able to hand-draw diagrams to be formalized in the tool, modeling with the voice, or modeling with a text editor.

15 out of 16 practitioners imagined features related to *domain knowledge* and *notation*. This covers the two most popular features of the list about *model completeness* and *model correctness*. 11 participants imagined that the ideal modeling assistant could be able to detect what is missing from a model, and to add new model elements in an auto-completion manner. For example, participant P4 would like the system to be able to "*complete my model and offer the possibility of making choices, i.e., to choose whether or not to include this concept, in fact to add and take on aspects that we quickly forget, security aspects, because behind it all we come with a particular background, we are never omniscient about all aspects and so at times when we are going to model, we will have a particular concern in mind. The assistant must therefore be there to enable us not to forget the really pure IT type aspects, the security type, the data type, the type of how I can make my application more resilient*".

In a similar fashion, 10 participants would like their modeling assistant to be able to detect model inconsistencies, to assess and validate the model and, if necessary, suggest how to fix such errors. Both features provide assistance about the domain by suggesting elements or assessing rules that practitioners might have forgotten, while helping on coping with the notation by preventing the misuse of incompatible syntax elements. Some participants also imagine the assistant as a peer, which is capable of advising them on best practices (4 out of 16), or answering questions on the business domain (2 out of 16).

10 participants require assistance on the *corporate methodology*. The features that fall into this category profile an assistant capable of providing active help with internal methodology practices (4 out of 16), the choice of model elements to create (3 out of 16), or the choice of the diagram to create (3 out of 16).

12 participants out of 16 also think that a digital modeling assistant could be useful when working along with the client in a meeting or a live modeling session. Table 4.5 presents the features that they would like to use in such situations. Firstly, practitioners would like to use different ways to interact with the tools such as speech-driven modeling (4 out of 16) or hand drawing on whiteboards (2 out of 16). Secondly, they

| Feature | # | Domain knowledge | Corporate methodology | Notation | Tool usage |
|---|---|---|---|---|---|
| Complete/detect what is missing in the model | 11 | X | | X | |
| Validate/detect what is wrong with the model | 10 | X | | X | |
| Generate/make diagrams for me | 6 | | | | X |
| Prepare diagrams (with templates) | 5 | | | | X |
| Layout the diagram autonomously | 4 | | | | X |
| Ensure consistency between multiple diagrams/models | 4 | | | | X |
| Advise on good practices | 4 | X | X | X | |
| Adapt menus and options to the current context | 3 | | X | X | X |
| Suggest what type of diagram to make | 3 | | X | X | X |
| Answer questions about the domain | 2 | X | | | |
| Recognize and formalize hand-drawn models | 1 | | | | X |
| Propose layers for the objects in the diagram | 1 | | | | X |
| Support speech-driven modeling | 1 | | | | X |
| Modeling by writing wysiwyg text | 1 | | | | X |
| Advise on business process modeling order | 1 | X | X | | |
| Suggest resources to read or people to contact | 1 | X | X | | |
| Repair models | 1 | | | X | |
| Suggest ways to better use the too | 1 | | | | X |
| Adapt the precision of the tool to my level of modeling | 1 | | | | X |
| Evaluate the impact of a change on a model | 1 | | | | X |
| Draw informal diagrams in the modeling tool | 1 | | | | X |
| Total of respondents | 16 | 15 | 10 | 15 | 16 |

Table 4.4 – Participants' features for their ideal modeling assistant

would like the assistant to guide the client when conceiving the system with questions, or steps to follow (2 out of 16). Finally, participants also would like the system to listen to the conversation, and be able to generate reports about the meeting (2 out of 16), explain previous design decisions (1 out of 16), or provide metrics about the evolution of the model (1 out of 16).

| Feature | # of participants |
|---|---|
| Support speech-driven modeling | 4 |
| Recognize and formalize hand-drawn models | 2 |
| Generate reports about the meeting | 2 |
| Guide the client with questions or steps to follow | 2 |
| Explain design choices made in previous meetings | 1 |
| Identify wrong design decisions | 1 |
| Propose a diagram template based on a general topic | 1 |
| Layout the diagram quickly | 1 |
| Generate metrics about the evolution of the model | 1 |

Table 4.5 – Features for the ideal modeling assistant with a client

### R.Q. 2.3: What makes a good software modeling assistant?

Questions B8 and B9 were designed to identify aspects that may influence how participants perceive the assistant. Practitioners first listed

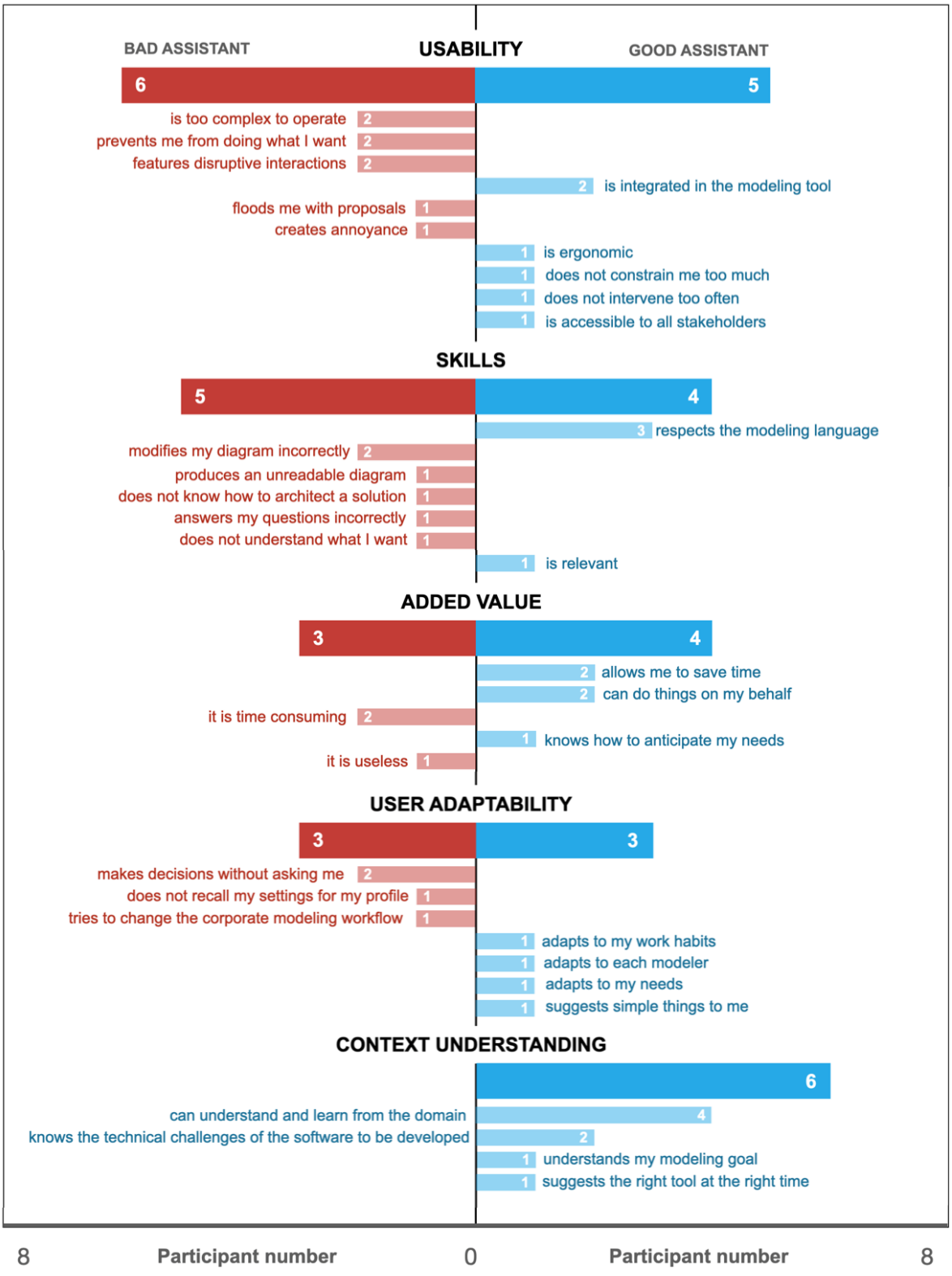



Figure 4.7 – Participants' answers to what makes a good or bad software modeling assistant.

what makes the assistant good for them, and then had to list characteristics (different from those already listed) that make the assistant bad. We

note that these characteristics had to be imagined and applied to a modeling assistant that does not exist. Thus, the practitioners' ideas reflect the features they thought of, and not necessarily the reality of practice. Figure 4.7 presents the 33 features mentioned by the participants, which we have grouped into 5 categories: usability, skills, added value, user adaptability, and context understanding.

The usability of the modeling assistant influences the perceived quality of the modeling assistant for 9 participants. This includes the way the software is used but also its behavior towards the user. 9 participants think that the skills of the assistant influence its quality, such as if it makes mistakes, misunderstands the user, or does not respect the modeling language. 5 participants care about the added value of the assistant to judge if the system is good or bad. Their reflection is mainly about saving time and taking over tasks for them. The fact of adapting its behaviour to the user is important for 5 practitioners, for whom the assistant must conform to their way of working, include them in the decision process, but above all adapt to their habits and needs. Finally, for 6 participants, a good assistant must understand the work context, learn from the domain, understand the issues, and understand what the user wants to do.

### 4.2.4 R.Q. 3: How would software modeling experts like to interact with the ideal software modeling assistant?

The modeling assistant is a system that aims to help users, by advising them about things, doing tasks for them, or both. To do so, it must take part in the overall task at a specific moment, and might identify things to do, how to do them, make decisions, and apply actions if necessary. For all these subtasks, the system can have different levels of automation, ranging from non-automated (not helping the user), to fully automated (not involving the user in the task), separated by different intermediate levels of automation [130]. Among these intermediate levels there are those where the system proposes alternatives among which the user must choose. In this section, we report on questions C2 to C11, which aim to identify the desired level of automation as well as to understand how the participants would like to interact and collaborate with the modeling assistant. Figure 4.8 introduces the results for the five subquestions of this section.

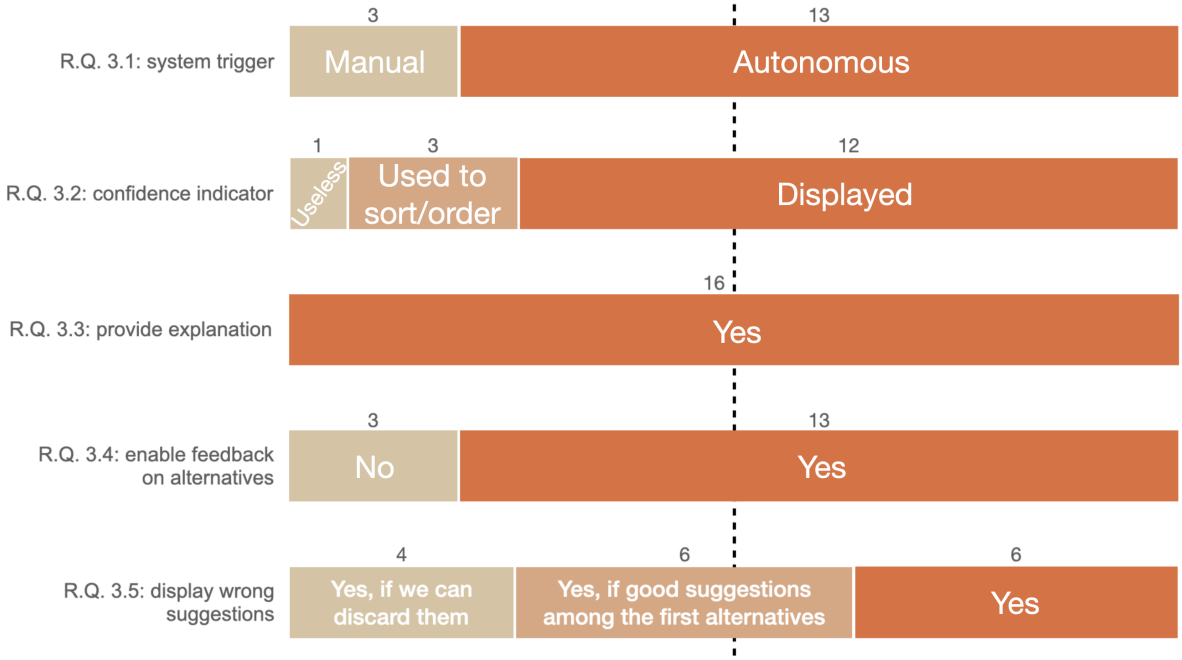



Figure 4.8 – Overview of the results of R.Q. 3 subquestions

### R.Q. 3.1: How should the system be triggered?

Questions C2 and C3 sought participants' opinions on how to activate the modeling assistant. 13 practitioners would like the system to be able to trigger itself without their explicit request, i.e., proactively. However, once triggered, the system should be controlled by the user and should not act on its own without prior validation. The 3 participants who want a reactive assistant, i.e., one that reacts to their request, do not give any justification except for their personal preference on how to interact. They declare that they *"hate systems that manifest themselves by saying what they should do"*. We note that 3 (different) participants took as a counter-example of good interaction Clippy, the famous Microsoft software assistant that appeared in the versions of Office from 1996 onwards. Clippy quickly became notorious for interrupting when it was not wanted; it failed to model user goals correctly, and it failed as a communicator [166, 18].

### R.Q. 3.2: Should the system indicate how confident it is?

Question C6 was intended to understand whether participants would like to have an indication of the confidence that a modeling assistant might place in its suggestions. The question was about whether or not to display a confidence indicator when making a suggestion. 12 participants would like to see a visual indication with the system's

suggestion that expresses the confidence level of the system, such as percentages or colors. In their opinion, this indication could increase the transparency of the system, and improve their use of the tool. Participant P8 states that *"if the tool is not certain to be able to do it right, I'd prefer it tells me it isn't sure it can do it. There's nothing worse than finding out later that the tool is really messing up"*. Participant P3 said that *"if it says I'm confident in 40 percent, then I'll know that every time I've used 40 percent, it's been wrong, so I won't take the 40 percent, I'll take another value. On the other hand, if each time it has made a proposal to me at 40 %, it has always had good advice, in that case I will take its proposal because I am quite confident about what it is capable of giving me. So it's also a learning process between the human and the machine on its level of confidence which corresponds to a level of confidence, in me, too"*. Of the 4 participants who did not wish to see the level of trust, 3 nuanced their remarks by indicating that even if they did not wish to see the level of confidence explicitly, the system should organize its proposals and/or actions according to it. A list of proposals should thus be sorted according to the level of confidence, even if it does not appear.

**R.Q. 3.3: Should the system explain its suggestions?**

In the case where the modeling assistant makes suggestions, 16 out of 16 participants would like to have an indication of why the suggestion was made. Four of them qualified their answer by indicating that this explanation should be provided on demand, if they decide to do so, but not automatically every time. Practitioners mention wanting this explanation in order to understand the origin of a proposal (7 out of 16), to learn from the assistant (5 out of 16), to evaluate and compare one's work to proposals (4 out of 16), to better choose a proposal (3 out of 16), and finally to identify the sources of information behind the proposal (1 out of 16). Participant P9 sees these explanations as a way to solidify the relationship of trust between the user and the machine, and finds it essential *"that it can 'justify' itself in the sense that for example it says "I understood your request in this way, I propose this, do you agree or not?", always with the idea of learning and enriching because I know that many users want to understand, have the hand on the output, or do not trust. So it is to reassure them, to perpetuate the action of the bot"*.

**R.Q. 3.4: Should the system provide a feedback mechanism?**

As discussed by participant P9 in the previous section, these practitioners may not agree with the suggestions of the assistant, which might not

be fitting their work context. Questions C10 and C11 ask whether in this case participants would like to be able to indicate their disagreement or simply ignore the incorrect suggestions. 13 participants indicated that they wanted a feedback mechanism to influence the system for learning purposes. The participant P5 even considers this feedback phase normal because *"it's personalization, that's the goal of these assistants, at the beginning we don't know each other and all the elements I'll be able to give it about my preferences or my needs, it will only improve the following because the choices it will give me will be conditioned by what I will have given it"*.

3 participants did not want to be able to give feedback on the proposals, each citing a different reason. A first practitioner thinks that the system should be able to learn by itself, another thinks that the repetition allows the practitioner to doubt the work in progress, and a last one simply thinks that it would not be useful.

**R.Q. 3.5: Could the system display wrong suggestions?**

When working with the practitioner, it is possible that the assistant makes bad suggestions that are not appropriate for the current time or context. Questions C8 and C9 address the acceptability of these incorrect suggestions. When asked, 16 out of 16 participants said they were willing to see incorrect suggestions, under certain conditions. Two of them would only want to see them on demand, and one would only want to see them depending on what stage of modeling they were in. Participant P13 discusses the inclusion of bad suggestions by saying that *"It's kind of like Netflix, sometimes I wish they would offer me things that are out of my habit and sometimes I'm really glad they do because I really want to see something in my habit so it's true that it's kind of complicated"*.

The conditions cited for being able to post bad alternatives primarily relate to the presence of a good alternative in the list of proposals. Thus, 2 participants want a good alternative among the first 3, 3 participants want one among the first 5, or 1 participant wants one among the first 10. Other participants (4 out of 16) do not focus on the number, but ask to be able to delete them, by giving their feedback, if they are displayed.

### 4.2.5 R.Q. 4: How could a software modeling assistant help experts' colleagues?

While in the previous research questions we investigate how participants would like to be assisted, participants sometimes have to act as assistants, especially with their colleagues. This section presents the results of the part D of the questionnaire, which focuses on the notion of assisting

colleagues. These questions aimed to understand if and how an assistant might help participants' colleagues, based on their experience.

### R.Q. 4.1: How are participants solicited by their colleagues to help them?

In response to question D1, 9 participants report being asked to do something at least once a month by their colleagues, 5 report being asked to do it several times a week, and 2 report being asked to do it every day.

Table 4.6 presents the general tasks for which participants are solicited. Six of the eight tasks identified can be classified in the five categories mentioned in Section 4.2.2, and correspond to tasks that can be performed alone. Indeed, 10 participants assist their colleagues on Domain Knowledge notions (it includes finding useful resources), 5 help on tasks intrinsic to modeling know-how, 3 answer questions on tool usage, 2 help with scoring, and 2 help with following the corporate methodology. Only the tasks of answering questions about a diagram that has been made (2 out of 16) and collaborative modeling (1 out of 16) cannot be done alone, and require the support of another actor.

| Task | # of participants |
|---|---|
| Understand domain concepts and processes, and how to model them | 9 |
| Structure the diagram | 5 |
| Use the modeling tool | 3 |
| Answer questions about a diagram a participant made | 2 |
| Use the modeling language | 2 |
| Follow the corporate modeling methodology | 2 |
| Do collaborative modeling | 1 |
| Find useful resources | 1 |

Table 4.6 – Tasks on which participants help colleagues

When called upon by someone making a model, 3 said they were called to correct the model or propose new solutions, 5 only to evaluate the model and give feedback, and 7 to do both depending on the time. Only one practitioner (P13) did not answer this question, as his colleagues never make models.

### R.Q. 4.2: How do participants feel about their colleagues' trust in their answers?

Questions D6 and D7 identified that participants mainly feel that their colleagues trust their answers. When asked why colleagues use them,

practitioners identified five areas that they felt created this trusting relationship. First, their high number of years of experience seems to be one of the main criteria for 10 participants. Their position in the company as a reference in modeling, as the sole modeler or as a supporter of the modeling process is also a reason for 6 practitioners. 2 practitioners believe that their reliability (they do not leave the colleague without anything, and will help them to have a solution in the end) contributes to establishing the trust relationship. The way they work, by projecting themselves as non-experts, by challenging the colleague without giving them a ready-made solution, or by proposing new ideas out of the box, is a reason for their solicitation for 2 practitioners. Finally, human qualities such as openness, availability, or reactivity are criteria that play a role in solicitation for 1 practitioner.

**R.Q. 4.3: Could a digital assistance system for modeling act like participants to help colleagues?**

After the participants clarified how they help their colleagues in section D of the interview, the rest of the questionnaire was designed to understand if they thought a modeling assistant could help their colleagues in the same way they do. 8 out of 16 participants think this possible, and 2 more participants think that is possible, but not as well as they do.

15 out of 16 practitioners, however, identified features that could be implemented to help their colleagues. This difference means that participants feel that today, software is unable to help their colleague as they think it can. The identified features are presented in Table 4.7. It can be seen that 9 out of 12 features were already mentioned in Table 4.4 which presented the features they hoped for in their own software assistant. The new functionalities concern the presentation and explanation of example diagrams, and model instantiation to understand how it works and generate metrics.

**R.Q. 5: What limitations could prevent the development of the ideal software modeling assistant?**

During the interview, some participants were cautious about the possibility of creating the modeling assistant they imagined. This is notably the case for question B7 presented in Section 4.2.3, where 5 practitioners question the possibility of developing a wizard that could help them with the issues they have raised. Question D8, presented in Section 4.2.5, also highlights that 6 participants believe that it is impossible for a software assistant to help their colleagues on the same issues that they

| Feature | # | Already identified |
|---|---|---|
| Validate/detect what is wrong with the model | 7 | X |
| Complete/detect what is missing in the model | 7 | X |
| Guide on the corporate modeling methodology | 2 | X |
| Prepare diagrams (with templates) | 2 | X |
| Suggest resources to read or people to contact | 2 | X |
| Explain examples of diagrams | 2 | |
| Instantiate the model to obtain metrics | 2 | |
| Help with the use of the modeling language | 1 | X |
| Suggest ways to better use the tool | 1 | X |
| Layout the diagram autonomously | 1 | X |
| Answer questions about the domain | 1 | X |
| Suggest diagrams that might interest practitioners | 1 | |

Table 4.7 – Features to assist participants' colleagues

do. One participant mentions the maturity of the technologies, and in particular that *"to imagine that an assistant can, all by itself, create the content of the model, I think we are looking at least 10 or 15 years ahead"*.

For all of these participants, the problem cited is business expertise, including understanding the customer's needs and the business ecosystem. They believe that *"this is information that only the customer has"*, that *"The only person who could help me is the customer because it's not technical"*, and that *"the main problem is that the assistant doesn't have the knowledge of the customer's need so it won't be able to propose"*. According to these participants, this problem of acquiring the customer need is the main blocking point to designing a modeling assistant. The aspect of integrating the solution into the business ecosystem is also addressed as a limitation *"because the questions are functional, that is, often it is not a question about the substance of the schema, it is a question between the ecosystem and the schemas"*. One participant insists that *"on business analysis, as I do it, I find it very hard to believe that an assistant would be able to help me"*.

The results from all the questionnaire entries enable us to draw an overview of the practitioners opinions and work habits. In the next section, we build upon the presented results to provide a discussion about the reasons and the consequences of our findings.

## 4.3 Discussion

The aim of this section is to analyse the results, compare our observations to the literature, and identify trends or lines of research to explore. We present this discussion in the order of the research questions presented in Section 4.1.1.

### 4.3.1 R.Q. 1: Does the profile of the participants meet the profile of software modellers as presented in the literature?

The profile of the respondents indicates that they are senior modelers, practicing formal and informal modeling on a regular basis, and therefore have experience with recurring field issues. Our sample of participants mainly uses modeling at work as a communication and documentation tool, but not in a model-centered approach like MDE or MBSE. Indeed, our participants mostly practice UML modeling in a limited formalism, adapted to their needs. When asked about formal modeling, the majority of them mention numerous drawing tools such as Microsoft Powerpoint or Microsoft Visio, which reflects their lack of interest in strict compliance with the UML standard. We also note that, given the number of modeling tools mentioned, there does not seem to be a consensus on the use of a tool in the industry. The choice of the tool, most often imposed by the company, poses first of all interoperability problems well known by the community. But this great diversity of tools also complicates the modeling task for the engineer, who must learn to use a new software before working in a new company. Our participants' responses are representative of the known use of UML diagram types, sharing the 5 most used diagram types with the Hustchinson et al. MDE study [77] and the Petre UML study [136].

Participants also identified items that they find complicated during the modeling process. These are not necessarily tasks that they have difficulty with, but only tasks that carry the complexity of modeling in their opinion. We cross-referenced this list with the list of potential difficulties with modeling from the Forward et al. study [58]. Important overlapping problems include keeping models up to date, the complexity and cumbersomeness of modeling tools, and the use of UML. Our study also highlights additional issues, first of all related to the essential complexity of the modeling work. The activity of adapting the content of the model to the profile and skills of the recipient in order to ensure that it is understood is identified by the most participants as a complicated

point. This can be explained by the modeling usage profile of our panel and the culture of their company. Our participants mainly use modeling in a non-formal way, in companies where the modeling culture is not very strong. Indeed, not all stakeholders can read and understand diagrams in the same way. Thus, special care must be taken to create a diagram that can be useful and usable for the targeted parties. The other main emerging issue is business domain knowledge. This is distilled in almost all the answers of our interviewees and takes a central place in the subject of software modeling assistance. It will be discussed in more detail in the following section.

Participants range in age from 31 to 56, work in a variety of sectors, for companies of various sizes, and thus create an evenly distributed panel of respondents. Furthermore, our results indicate that the profile of our sample of practitioners seems consistent with the literature about UML and modeling tool users. Thus, the convergence of ideas between several participants appear as a criterion that suggests their potential generalizability. In the remainder of this section, we will formulate several hypotheses, based on the convergent ideas of our panel. These hypotheses call for further work to study their validity on larger samples, in a controlled environment, during quantitative experiments.

### 4.3.2 R.Q. 2: What do software modeling experts expect from the ideal software modeling assistant?

It appears that modeling practitioners expect assistance. Although only 9 out of 16 call upon the help of another collaborator on a regular basis, all of our participants identified features that they would like to be assisted with in their work. Since these features were imagined as part of a software system, this indicates that participants project themselves into using a software assistant in their daily work. In general, the respondents showed a very strong attraction to a software assistant that could help them during the interviews. We thus formulate Hypothesis 1.

**Hypothesis 1**
*Modeling experts want to be assisted in their work. They are ready to welcome a software modeling assistant for this purpose.*

The 22 functionalities identified in Table 4.4 provide research and development avenues for researchers and companies responsible for modeling tools. These results do not indicate that a super-assistant

should have this many features, but rather open the door to the creation of multiple assistants to address one or more of the features identified. 100% of the practitioners would like to be assisted in the use of the tool, an issue well known to the modeling community. Assistance in the use of the modeling language is also mentioned by 15 of the 16 participants, which is in line with several studies on UML. This can be explained by the frequent use of less formal drawing tools, which do not impose any constraint on the use of a modeling language, leaving the user to choose the concepts to be created and connected in the diagram. The freedom thus offered by the modeling tool can appear as a setback when it comes to respecting a syntax or a meta model in a more formal way. 10 participants ask for less freedom when it comes to respecting the enterprise modeling methodology. In particular, they would like to benefit from methodological guidance in the tool, allowing them to apply the internal best practices and the business modeling process in the best possible way.

The two most mentioned functionalities are centered around business domain knowledge. This overlaps with the main aspect on which practitioners call for external help when modeling. A total of 13 practitioners mention an intelligent model analysis system that produces metrics or reports on what is missing or invalid in the model. These reports would be generated using the system's internal knowledge, which could compare their work to its knowledge base. Following the reading of these reports, these participants would like to have the possibility to act accordingly, and to benefit from systems able to suggest how to complete or correct their models. Thus, we formulate hypotheses 2 and 3.

**Hypothesis 2**
*Modeling experts want to have analysis reports about their models. These could include what is missing in the model, but also what is functionally or syntactically inconsistent.*

**Hypothesis 3**
*Modeling experts want to benefit from modeling recommendation systems, suggesting new elements to add to the model, or corrections to apply to the model.*

We also note that regarding the business domain, participants mentioned functionalities that are not directly related to a modeling task, but

rather to background tasks related to the acquisition of knowledge allowing to model. These tasks, such as advising on best practices, answering business questions, or suggesting resources to read or people to contact, are usually performed outside the modeling tool. These issues are often time consuming, and require the intervention of other stakeholders to be resolved. The identification of such tasks by the participants suggests that the disruptive interaction they impose in their current form hinders the practitioners, and leads us to the formulation of hypothesis 4.

**Hypothesis 4**
*Modeling experts want to be able to access business information more easily, without having to rely on external stakeholders in a systematic way. This information could be integrated into their modeling environment.*

We propose in Figure 4.7 the exhaustive list of the criteria of good or bad assistant as evoked by the participants. Although not all the criteria are precise, and some are too general to be useful, this grid can be used as a first approach to evaluate the quality of a product assistant. By applying these remarks to their prototype or system, researchers or industrialists can identify areas for improvement regarding the users' perception of the usability of the assistant, its skills, its added value, its adaptability to the user, and its understanding of the context. The notions of usability and user adaptability concern the human-machine interaction aspects of the assistant, whereas skills, added value and contextual understanding refer more to the knowledge and content of the assistant. Modeling experts want to be able to access business information more easily, without having to rely on external stakeholders in a systematic way. This information could be integrated into their modeling environment.

**Hypothesis 5**
*The interactions and behavior of the assistant, as well as the knowledge and skills he or she holds, each influence the perceived quality of the assistant. A good assistant fulfills criteria of usability, competence, added value, adaptation to the user, and understanding of the work context*

### 4.3.3 R.Q. 3: How would software modeling experts like to interact with the ideal software modeling assistant?

Building the most accurate system may not be the top priority when dealing with software modeling assistance. While much research measures the accuracy of assistants, especially recommendation systems, our results indicate that other criteria should be studied first to maximize the acceptability and usability of the systems created. Indeed, 16 of the 16 practitioners we interviewed are willing to be suggested bad alternatives, as long as the system's behavior allows them to refute them, or to find good ones quickly. These aspects of behavior, reflecting the way the user interacts with the system, seem to take priority over accuracy when practitioners describe the ideal system. The lack of consideration of these user interface issues seems to explain why so few recommendation systems for software engineering have been adopted so far [117]. This observation is corroborated by hypothesis 5, which emphasizes the notion of human-computer interaction in what makes a good modeling assistant.

The answers of our participants, almost unanimously, enable us to draw the big picture of what they expect in terms of interaction about distraction, scrutability, transparency, and assessability [117, 182]. These criteria allow the establishment of trust between the recommender system and the user, which promotes usability but also the perception of the quality of the system [182]. Applying these criteria could therefore promote the adoption of the concerned modeling assistant.

**Hypothesis 6 (Distraction)**
*The modeling expert assistant should be able to manifest itself, offering a non-disruptive interaction such as a notification or annotation. This feature must be able to be disabled, and the assistant triggered on demand.*

**Hypothesis 7 (Understandability and transparency)**
*The modeling expert assistant shall order its proposals according to a confidence score, which shall be displayed. The confidence score should be hidden on demand.*

**Hypothesis 8 (Assessability and transparency)**
*The modeling expert assistant must be able to explain why a proposal was made. This explanation must be able to be displayed either all the time or only on demand.*

**Hypothesis 9 (Scrutability and control)**
*The modeling expert assistant must allow the user to refute its proposals, and must be able to learn about the user's needs based on these interactions.*

### 4.3.4 R.Q. 4: How could a software modeling assistant help experts' colleagues?

All the participants indicated that they help their colleagues at least once a month, by acting as an assistant to evaluate or correct the work presented. It appears from the interviews that the majority of the assisted colleagues are less experienced than our participants, and can therefore be considered as average (non-expert) software engineers. Our practitioners seem to think that their colleagues are confident in their answers mainly because of their high number of years of experience. Knowing that the correction or suggestion originates from an experienced practitioner seems to influence the confidence the respondent has in the proposal. This is in line with the results of Chopra and Wallace [41] which indicates that displaying the source of information participates in the construction of information trustworthiness.

**Hypothesis 10 (Trustworthiness)**
*The modeling assistant must indicate the sources of information used to create the proposed alternatives.*

Although half of the practitioners interviewed felt that an assistant would not be able to help their colleagues as they do, they profiled the assistant that could still help them. Table 4.7 shows that 9 of the 12 features identified are identical to those mentioned for the participants' modeling assistant. Similarly, the majority of tasks on which participants help their colleagues (shown in Table 4.6) are similar to those where participants need help (shown in Table 4.2). These results suggest the generalizability of our findings to a broader portion of the modeling

community. However, participants identify the need to present more examples, and to further explain the diagrams to their less experienced colleagues. We thus formulate hypotheses 11 and 12.

**Hypothesis 11 (Generalisability)**
*The modeling wizard for experts can also be useful for beginner and intermediate software engineers practicing modeling.*

**Hypothesis 12 (Examples and explanations)**
*The modeling assistant for junior and intermediate software engineers needs to provide more explanations and examples.*

**R.Q. 5: What limitations could prevent the development of the ideal software modeling assistant?**

A significant part of the participants indicated their skepticism about the possibility of creating the assistant they imagined. This feeling is mainly justified by the fact that, according to them, the assistant could (i) never have the same business knowledge as the customer and (ii) never have the knowledge of the entire ecosystem in which the new solution must be integrated. This remark raises different points concerning knowledge acquisition and indicates possible avenues to explore to improve the potential of the assistant. First of all, even if the assistant has knowledge about the business domain of the solution to be implemented, only the customer is the stakeholder aware of the real need that justifies the implementation of the new system. In fact, one way to increase the potential of the assistant would be to involve the customer in the operation of the assistant. This could be done by using technologies already explored in research such as the analysis of software specifications [1], the use of chatbots [51] gathering knowledge from customer, or by listening and analyzing conversations during meetings with the customer as discussed by our participants (see Section 4.2.3). The solution must also be integrated into an ecosystem with its technical and functional constraints, and its specificities. The new system must then comply with these specifications, which often go beyond the framework of the project and apply to the entire company. In the same way, the use of specification analysis and documentation techniques internal to the company could be a first step towards understanding this global context. However, as mentioned by our participants, there is often a gap between

the documentation of business processes and reality. In this case, the use of probes and bugs at key stages in the company, coupled with big data analysis techniques, could allow the collection of data that could be exploited by the assistant to refine its knowledge of the company.

Alongside the problem of data collection, there is also the problem of the maturity of the technologies that will enable the expected functionalities. Participants mentioned a set of functionalities ranging from the implementation of new interaction methods to the intelligent analysis of models, leading to advice and recommendations. Even if some of these functionalities have already been discussed in the scientific literature [12, 77], almost none of them is industrialized and available in a modeling tool to our knowledge. More generally, the participants evoke an intelligent system able to learn from their behavior, from their feedback, to give explanations for its proposals, and to intervene at the right time without disturbing the user. The competition of these functionalities raises problems, such as the general topic of explainability of artificial intelligence systems, or the design of interfaces of recommendation systems for software engineering. Finally, the results of Section 4.2.2 show the great diversity of tools used in companies. Creating a system that is both available to the largest number of practitioners and integrated with the modeling tool (mentioned in the points making the system a good assistant) would imply developing a system for many platforms, sometimes inaccessible because private.

## 4.4 Threats to validity

In this section, we outline the main threats related to the validity of our study, and the steps we took to mitigate them.

**Size of the sample.** The work reported here is indicative, resulting from a first qualitative research effort to understand software modeling experts better. It identifies and confirms trends and challenges in the practice of modeling, and paves the way to the understanding of the concept of engineer assistance in software modeling. However, we acknowledge that our participant sample is not large enough to claim comprehensiveness, as our results might exclude potential ideas and challenges. We attempted to mitigate this threat to validity by gathering participants from a broad range of small and large companies, business domains, and years of experience. We also restricted the sampling to one participant per company, in order not to influence the results about in-organization practices. Our results for research question 1 suggest that our sample is representative of practitioner populations studied in

other works, which may also help mitigate this threat.

**Researcher bias.** Researcher bias refers to any kind of negative influence of the researchers' knowledge, or assumptions, of the study, including the influence of their assumptions of the design, analysis, or sampling strategy. This threat applies particularly to our interview-centered approach, during which the participant and the researcher are brought to exchange and react to each other's words. In order to limit the influence of an external opinion as much as possible during the session, the interviewer had to follow the question protocol as presented in Table 4.8, where all interventions were defined beforehand. In some cases, the researcher could use prompts to clarify the participant's opinion, without expressing or introducing new vocabulary words. In order to further mitigate this threat, only one researcher conducted the entire interviews, to prevent the way questions were asked from becoming influencing factors.

**Respondant bias.** This threat refers to a situation where respondents do not provide honest responses for any reason, which may include them perceiving a given topic as a threat, or them being willing to please the researcher with responses they believe are desirable. In the call for participation to our study, participants were provided with a 2-line abstract of our research works and a website of one of our project about models and artificial intelligence. This information was necessary to give credibility to our experiment and to encourage participants to get involved. However, this was the only information provided to the user beforehand, which does not mention any opinion on the issues discussed in the session. The interviews were confidential, which allowed the participants to talk about any topic without limits.

In addition, we designed the questionnaire to focus on the tasks and avoid value judgments about the participants, in order to limit self-evaluation bias. We also mixed the style of questions (open-ended, closed-ended, fill-in-the-blank, to-do lists) to avoid participant boredom over the duration of the interview. The oral interview format helped to limit the interpretation bias of the questions, by being able to clarify or rephrase the questions when a participant did not understand. The order of the questions could not be randomized, so as not to bias the participant's answers, especially about the notion of assistance.

**Confirmation bias.** This threat relates to researchers interpreting the data to support their hypothesis, by potentially omitting data that does not favour their hypothesis. In this study, we recorded the interview sessions and had them professionally transcribed to text. Then, we manually coded each text file with a dedicated tool, that enables to

identify non-coded text portions, and to create reusable codes for each question. In this chapter, we present all results that were coded in the project, and thus mitigate the confirmation bias.

## 4.5 Conclusion

The main purpose of this study was to identify and characterize the need for assistance of modeling practitioners. Our results seem to indicate the strong interest of practitioners towards software systems able to help them, which we call software modeling assistants. The answers to the different research questions allow us to propose a set of requirements that the implementation of a software modeling assistant should satisfy, in terms of functionalities, behavior, and interactions.

The ideas discussed in this chapter are a step forward towards solving the issues related to modeling tools, that are well known in the community.

These preliminary guidelines can be used by the editors of modeling tools, but also by anyone who would like to develop complementary systems to these tools. They are a first contribution to the field of modeling assistance, and need to be studied in more detail, especially in empirical user experiments to evaluate them. In order to generalize these results, a shorter questionnaire, based on the formulated hypotheses, can be conducted on a larger scale online, in a quantitative research effort. Our work paves the way for a more detailed study of the modeling practitioner population, in order to understand their needs, and to be able to address the challenges they face. In the following chapter, we confront our results from both a systematic mapping study and these interviews, to build the big picture of our vision of modeling assistance.

| # | Question |
|---|---|
| | I am going to ask you some open-ended questions about modeling. The goal is to improve our knowledge of how you model and work. I invite you to take your time to answer. No aspect of your work is obvious, and pretend I don't know anything about the different tasks you talk about. If a question I ask you is not clear, feel free to ask for a rephrasing. |
| A1 | Can you briefly introduce yourself, talking about your education, your career and your different positions. |
| A2 | What is your current position and responsibilities? |
| A3 | How long have you been doing software modeling? |
| A4 | From 1 to 5, how would you rate your mastery of UML? |
| A5 | To start, let's get some context. Tell me about one of your last and most significant modeling experiences. Tell me from start to finish, how it started, everything you had to do, with whom, with what, in a chronological way. How it starts, and then..." I'm going to ask you some questions to clarify. They may repeat information that you have already given, so please feel free to elaborate. |
| A6 | Can you summarize the representative workflow of your general modeling work? |
| A7 | What kind of diagrams/models do you make? |
| A8 | For what purpose do you model? What do you do with these models? |
| A9 | What modeling language(s) do you use? |
| A10 | How often do you make diagrams informally? |
| A11 | When you do, what media do you use (paper, board, computer) to model? |
| A12 | Who do you collaborate with to model? |
| A13 | What means of discussion do you use (in person, by e-mail...)? |
| A14 | What kind of situation are you in when you start modeling? At what point in your project, in your work? |
| A15 | How often do you model for areas you know well? |
| A16 | Are you modeling for a client? If so, do you discuss with the client before modeling or during the modeling process? |
| A17 | What are the differences in your workflow between a project where you were super comfortable and another where you were much less so? |
| B1 | How often do you practice formal software modeling? |
| B2 | What tools do you use to model? |

| | |
|---|---|
| B3 | Why these tools? |
| B4 | Complete this sentence: the most complicated points for me during modeling are: ... |
| B5 | How often do you need help with modeling? |
| B6 | What do you need help with? |
| B7 | Do you think a software assistant could provide this help? |
| B8 | Complete the sentence: to help me model, an assistant is good if: ... |
| B9 | Complete the sentence: to help me model, an assistant is bad if: ... |
| B10 | Imagine that an assistant exists to help you model. In the ideal, best of all worlds, what should it be able to do? |
| B11 | If you had to choose three features, what features should a software assistant implement to help you? List them in descending order |
| C1 | Imagine the interface of your modeling tool and imagine that [FUNCTIONALITY 1] is available in the tool. Describe precisely how you would like to use it, each step to do in the tool, each click, on which element... Imagine you are using it, describe how it happens. |
| | (Do it for the 3 features) |
| C2 | In these features, is it up to you to request for assistance or for the assistant to show up by itself? |
| C3 | In your opinion, to what extent should a software assistant be able to manifest itself to offer you help? |
| C4 | Do you think this is feasible? Why or why not? |
| C5 | If a software assistant were to help you with business domain issues during modeling, how could this be done? |
| C6 | Imagine an assistant that makes several proposals to answer a problem. The assistant offers you one or more alternatives. To what extent should it, if at all, indicate the level of confidence it has in these alternatives? Why or why not? In what way? |
| C7 | Would you like it to explain why an alternative is being proposed? How would you like it? |
| C8 | Let's say an assistant offers you several alternatives for an action. What percentage of the alternatives that are not suitable for you at all would you accept to see?" |
| C9 | What are the conditions that would allow the system to display alternatives that are not suitable for you? |
| C10 | To what extent would you like to be able to indicate that some alternatives do not suit you at all? |

| | |
|---|---|
| C11 | How would you like to be able to indicate that some alternatives are not suitable for you at all? |
| D1 | How often are you asked to answer questions from your colleagues about modeling? |
| D2 | What modeling elements are these questions about? |
| D3 | How are you solicited? (verbally, by email, etc...) |
| D4 | Is it to evaluate their solution or propose new solutions? |
| D5 | How do you answer questions? (call, email, links, etc...) |
| D6 | Do you think your colleagues trust your answers? Why do they call on you? |
| D7 | Do your colleagues ask you to justify your answers with ""Why?""? How do you do it? What do you think is the reason?" |
| D8 | To what extent do you think a Software Assistant could do the job you are asked to do? |
| D9 | What would be the necessary conditions / limitations? |
| D10 | If you had to choose three, what features should a Software Assistant implement to help your colleagues? |
| D11 | Do you think an assistant could be useful during meetings with the client? What could it do? |
| E1 | Thank you for your answers, is there anything you would like to add? Is there anything else you want to talk about or do you have any questions? |

Table 4.8 – Interview questionnaire

Chapter 5

# The big picture of software modeling assistance

*This chapter draws upon the results of the three previous chapters to depict the current state of practice of software modeling assistants. It compares the observations of the literature with those of our interviews conducted with modeling professionals, highlighting the breaks but also the synergies between expectations and practice. From this analysis, we identify the key notions of modeling assistance, which should then be applied to the design of modeling software assistants, the next step of our user-centered approach.*

## 5.1 Expectations vs. Reality

**The need for assistance in software modeling.** The interviews conducted with 16 software modeling experts seem to confirm the strong interest of modeling practitioners towards software systems able to help them in their work, as stated by hypothesis 1 from Chapter 4. Indeed, 16 practitioners identified features on which they would like to be assisted, and a mean of 12 of them think a software assistant could actually help them on these features in the current state of technology and data availability. However, our mapping study of software assistants in software engineering only revealed 2 systems addressing modeling assistance in the literature. This shows a clear break between the expectations of modeling engineers, and the response of research to understand how to assist the modeling task in all its specificity and provide prototypes to

evaluate their approach.

In a pragmatic perspective, it is possible that this lack of research effort may be justified by the low profitability of the work produced. Thus, the justification would be that research in this field "does not pay". However, working groups, such as that of Mussbacher et al. [119] in 2020, have indicated the need to conduct work in this direction. It is then possible that the problem is not about the motivation of researchers, but about the availability of resources to conduct such research, a common problem in the academic world. In this case, these problems could also be of interest to the editors of modeling tools who are concerned with improving their tools to handle concrete problems encountered by practitioners in companies.

**Types of assistants.** The mapping study presented in Chapter 3 of the thesis introduced 3 types of assistants: *Informers* (10 out of 47, 21%), *Passive Recommenders* (21 out of 47, 45%), and *Active Recommenders* (16 out of 47, 34%). This appears mainly inline with hypotheses 2 and 3 from Chapter 4, resulting from our interviews. Such systems can then be able to provide analysis reports with metrics (informers), but also recommendations about what to fix or add in the model (passive and active recommender systems). Nevertheless, this analysis must be nuanced given the low proportion of assistants dedicated to modeling. While software assistants exist in software engineering to meet this kind of needs, the creation of such systems for modeling remains anecdotal.

**Human-Computer Interactions (HCI) indicators.** The largest discrepancy between the literature and the interview results was in the HCI indicators. While 100% of the 16 practitioners interviewed stated that they would like an explanation of why the alternatives of a recommender are suggested, only 1 assistant in the literature offers this service (less than 3 percent). Practitioners justify that such explanations would impact their creativity while enabling them to learn. Indeed, based on the explanations, one recommendation that was not likely to fit their needs at first glance could reveal its potential. It may open the way for new ideas and directions to explore, and it is why 16 out of the 16 practitioners would accept to see *bad* recommendations. Bad recommendations are bad from one perspective, and might fit the project when following another design approach.

Similarly, 81% of the practitioners (13 or of 16) would like to be able to give feedback to the system to make it learn or refute proposals, whereas only 3 out of 37 systems (8%) allow this in the literature. Finally, while 15 out of the 16 practitioners expect the system to use a confidence indicator to sort the recommendations, and 12 of them want to see it

explicitly, only 10 systems out of 37 implement it in the literature (27%). Feedback and confidence indicators both relate to the notion of trust, as stated by the practitioners. These elements could indeed help them develop a better relationship with the assistant by tailoring it to their needs, but also to understand better its behaviour.

**Trigger.** From the interviews, we learned that the majority of practitioners would like the assistant to trigger itself automatically but in a non-disruptive way. In the mapping study, we noted that 23% of Active Recommender Systems are triggered automatically, compared to 50% of Informer Systems and Passive Recommender Systems. In practice, as stated by hypothesis 6 from Chapter 4, modeling assistants should be able to manifest itself, as well as being triggered on demand. This emphasizes the need for thorough assistant interaction and automation design. The trigger but also the work scope of the assistant should be designed according to the goal of the system and the task of the user. If not, the user-assistant relationship might end up prematurely, as for Clippy, the Microsoft Office assistant [166, 18] which was jumping in at the wrong time, with incorrect predictions of users' intents.

## 5.2 Key notions in modeling assistance

The work carried out in the first part of this thesis, of (i) understanding the modeling task, (ii) studying the literature on software assistants in software engineering, and (iii) understanding the need of assistance of modeling engineers, allowed us to highlight the fundamental characteristics governing modeling assistance in a global way. In this section, we summarize the role of the four key notions of modeling assistance: *trust*, *creativity*, *recommendations*, and *automation*.

### 5.2.1 Trust

*Trust* was the most mentioned concept in the interviews with practitioners. Our conversations with these modeling experts highlighted the need to create a relationship of trust between them and the assistance actor in order for the latter to be effective. An established relationship of trust seems to make the software engineer accept bad recommendations more easily, for example by providing explanations. One possibility to foster trust appears to be through the transfer of authority from a colleague, if they have the status of an expert in the field. It also seems to be earned by accepting feedback if a proposal is not really relevant to the context.

In these cases, the software engineer seems to be able to modify the way their interlocutor thinks and proposes alternatives.

### 5.2.2 Creativity

The modeling task is a *creative task*, as explained in Chapter 2. Thus, taking part in the modeling process requires the ability to generate creative ideas. Practitioners mentioned this creative process when justifying why they would accept bad recommendations. Original ideas, unexpected propositions, or sometimes completely unrelated ideas have the power to stimulate thinking, and help new creative solutions to a problem emerge. At best, assistance actors should foster the software engineer's creativity when possible, to facilitate the generation of design ideas. In any case, the modeling assistance shall not prevent the expression of creativity.

### 5.2.3 Recommendations

Whether it is to mention what is wrong with the model, or to propose new solutions, modeling assistance involves *recommendations*. This can be recommending resources to read, people to contact, fixes, things to add, or even code to reread as potentially wrong or problematic. In all cases, data analysis is combined with knowledge to produce one or more alternatives, which are then ranked. The assistance actor then chooses to explicitly propose one or more suggestions, or to implicitly use them to take one or more actions. Hence, in order to provide modeling assistance, one must be able to provide recommendations.

### 5.2.4 Automation

In its broad definition, *automation* refers to the way in which help is requested, but also to the sequence of interaction between the helper and the helpee. It also takes into account the resources and information that will be shared during the help request. It is important that all of these aspects are clear before asking for help. For example, the software engineer may not want to share his or her confidential work with an outside party who cannot guarantee confidentiality. Similarly, it is unlikely that a modeler would seek help from a colleague who would completely change the layout of his work environment before providing assistance. In case of the interviews, practitioners mentioned Clippy the assistant as something not to commit again, and insisted on their preferences about triggers and interactions. Thus, modeling

assistance can only occur if the automation of that assistance is clearly communicated beforehand and adapted to the task the software engineer seeks to perform.

## 5.3 Towards formalizing modeling assistants

The above notions characterize modeling assistance from a global point of view. In particular, they apply to human assistance, which can be provided by interviewing practitioners and their colleagues. To design *software* assistants capable of providing such assistance, it is then necessary to investigate how these key notions can be applied in the software domain to create systems acting as peers. In the next part of the thesis, we propose a survey of the literature concerning the application of these four notions to software systems design, and put these results in perspective to propose an architectural framework for the design of software assistants for modeling.

# Part II

# Designing Software Assistants for Software Modeling

Chapter 6

# Identifying design constraints from the literature

*The first part of this thesis report highlighted that software assistants were candidate solutions to tackle modeling problems. Chapter 2 demonstrated that the modeling issues from the literature require knowledge and well-designed user interactions to be addressed, and that software assistants can provide these by nature. Chapter 3 then outlined that software assistants are rarely exploited to support modeling activities in software engineering. Nevertheless, results from Chapter 4 tended to indicate that modeling practitioners call for more software assistants to support them in their modeling tasks, provided that they respect design constraints. This call from end-users to be supported in their work emphasized the relevance of our user-centred approach to design modeling assistants. Chapter 5 outlined the road ahead for the reality of the implemented software assistants to catch up with users' expectations, and most importantly identified the four pillars on which modeling assistance must rely.*

*The identification of these notions covers the first two phases of our user-centred approach, about (i) understanding and specifying the context of use of the system, and (ii) specifying the user requirements. The next step of the process is (iii) to produce design solutions to meet user requirements, as defined in Section 1.3.1. However, as we conduct a* research *user-centred approach, it is necessary to produce solutions that are both tailored to end-*

*users' requirements but also inline with the state of the research.*

*The purpose of this second part is to bridge the second and third steps of our user-centred approach, with elements from the research literature. In this part, we present how the literature in computer and cognitive science guides the implementation of the four key notions of modeling assistance within* software *modeling assistants. This results in the identification of constraints from the literature to design software modeling assistants. From these design constraints, we finally propose a formal framework, to guide the design of software assistants for software modeling.*

*In this chapter, we identify the elements of the literature that allow us to define systems that promote (i) trust and (ii) user creativity. Then, we detail how to apply these notions to (iii) recommendation systems adapted to modeling, (iv) whose automation is consistent with the supported tasks.*

## 6.1 Enabling trust in human-assistant collaboration

### 6.1.1 The need for trust in modeling assistants

#### What is trust?

The Oxford dictionary [127] defines trust as the "*firm belief in the reliability, truth, or ability of someone or something; confidence or faith in a person or thing, or in an attribute of a person or thing*". Simon [169] defines trust as "*the belief that another individual, organization or institution will act in a manner consistent with what is expected of it*". In practice, trust governs most of our daily interactions, both with humans and with physical or software systems. It is also preponderant in the first impression that we make during our first interactions with these actors [183]. Trust cannot be forced or imposed: it is voluntary, spontaneous and natural; it is co-constructed in the bilateral relations that are established between individuals or between groups [11] It cannot also be produced, but results from the alignment of different factors in the relationship between the *trustor* and the *trustee*.

Maurel and Chebbi [109] state that three main types of trust emerge from the study of the literature. *Interpersonal trust* happens between individuals, regardless of their title or position. This trust has both affective and cognitive bases. *Organizational trust* occurs when members of an organization choose to engage in collective action as representatives of their respective organizations when more than one organization is involved (inter-organizational trust) or of their administrative unit within

the organization (intra-organizational trust). *Social* or *institutional trust* is based on a formal social structure, i.e., the legal and regulatory framework that governs each society, or the normative context of a discipline.

Trust is characterized by a dependency (sometimes reciprocal), as well as a certain vulnerability between the partners. One or more of these actors have expectations about the behavior of the others in order to establish a relationship of trust. These expectations may relate to the sharing of values common to individuals, social groups, or a company, but also to the objectives of the task to be carried out. The notion of trust is difficult to quantify - if it ever is. It is the result of a perception, and is therefore a subjective concept whose criteria are specific to individuals and their own characteristics. These are particularly based on cognitive factors, such as intelligence or knowledge, and conative factors, such as their personality traits. However, it seems that depending on the nature of the task and the actors, the notion of trust can be refined.

### Understanding trust for modeling assistants

The general definition of trust presented above is not applicable as such to software design. Even more precisely, the question of the definition of trust for the global domain of software engineering remains difficult to address. In this domain, two main types of relationships are to be considered: human-human and human-machine. Our work about software assistants only addresses the latter, and especially the relationship between the software engineer and their work tool in the context of a software modeling task. Although there are only a few studies about the general concept of trust in software engineering[1], many studies have aimed at answering the question of trust for more restricted domains. In particular, different studies discussed trust according to the type of software system considered. In their framework for trust in electronic environment, Chopra and Wallace [41] evoke trust criteria by distinguishing several types of systems such as information systems, e-commerce systems, or online dating systems.

A very limited research has been conducted to study trust for software assistants, if none. However, our vision of software assistants relies on gathering *knowledge* to be processed, adapted, and presented to the user as described in Section 2.3.1. This kind of software system aiming to provide the user with valuable information is regularly called

[1] In the last year, a lot of new efforts and projects to study trust in Software Engineering, such as trust for AI-based system started to emerge. However, this remains limited, compared to the state of the literature from the 20 last years.

*Information Systems*. As software assistants can be generalized to information system, we exploit the literature about trust for information systems. This is inline with the work of Boell and Cecez-Kecmanovic [23] who gathered 34 definitions of information systems and deeply analyzed their characteristics. Thus, according to their process view of an information system, a software assistant is "*a work system whose process and activities are devoted to processing information, that is, capturing, transmitting, storing, retrieving, manipulating, and displaying information*". From the technology view, a software assistant is "*a system that utilises computer hardware and software; manual procedures; models for analysis, planning, control and decision making; and a database.*" Consequently, fostering trust in software assistants can be achieved by exploiting the trust characteristics for information systems.

Previous research indicates that trust in technology is an essential antecedent to its adoption. [14]. This means that if designers are to make effective use of a recommender system (a software assistant is a recommender system in a way), they must have a sufficient degree of trust in it. However, the nature of the task to be assisted is key to the understanding of users' trust needs. Indeed, assistance systems (as information systems) can be deployed for a wide variety of applications, from the most critical to the most anecdotal. For example, a system that allows to decide what action to take to solve a critical problem of flying an airplane contrasts with a system that allows the recommendation of movies to watch on Netflix. In these two cases, the user's expectations are respectively robustness, accuracy and a very low error rate for the first one, against novelty, diversity, and a larger error tolerance for the second one. Our work focuses on assisting the modeling task, as described in Section 2.3.3. The nature of this task implies that the assistance system –the software assistant– is perceived as a bonus in the existing environments. Thus, the fulfillment of the modeling task does not rely on the availability of a software assistant, but might rather be facilitated when using one.

As a consequence, users do not have high expectations about the *availability*, the *robustness* or the *reliability* of the system, as they could have for critical systems. These previous characteristics are called *information system trustworthiness characteristics* by Chopra and Wallace [41], due their link with the system considered as a whole service. Instead, users expect the system to be able to come up with new, interesting and/or original ideas. Hence, when assisting modeling –a creative task– trust is mainly built on the *information* itself, and not on the *information system* as a whole software service. This statement implies that aspects

related to trust in information systems as described by [41], such as the availability, the robustness, the reliability, the safety of the system, or the system being free from malicious code, will *not* be considered as major factors influencing trust in modeling assistants and will not be further discussed in this section.

In their framework for trust in electronic environments, Chopra and Wallace define trust in information as *"the willingness to rely on a specific other, based on confidence that one's trust will lead to positive outcomes"*. Thus, we state that trust in modeling assistants will rely on the information these systems are able to provide. This includes the information itself, but also the mechanisms to adapt it to users' needs. The following section explores this direction, by providing a clear overview of how information-related and system adaptability characteristics influence the growth of trust.

Note that both *information* and *knowledge* are used in this section, referring to the same concept. Software assistants manipulate and produce *knowledge*; the use of *information* or *information systems* is constrained by the name of such systems in the community. An explanation about the difference between information and knowledge is provided in Section 2.2.2.

### 6.1.2 A metamodel of trust in recommender systems for creative tasks

Because recommender systems are information systems, we gathered articles linking both information and trust [41] [73] and cross-referenced them against literature pertaining to trust and evaluation of recommender systems [142] [182]. To analyse the contents of all these papers, we constructed a model (shown in Figure 6.1), which represents a consolidated conceptual model of trust in recommender systems.

#### Characterizing user's attitude towards the system

When using a recommender system, users develop a relationship with that system, and hence adopt a specific attitude towards it. Our study of the literature shows that this attitude is tight to different characteristics related to the user and influenced by the system.

- **Trust** indicates whether or not users find the whole system trustworthy.
- **Confidence/Persuasion** refers to the recommender's ability to convince users of the information or products recommended to them.

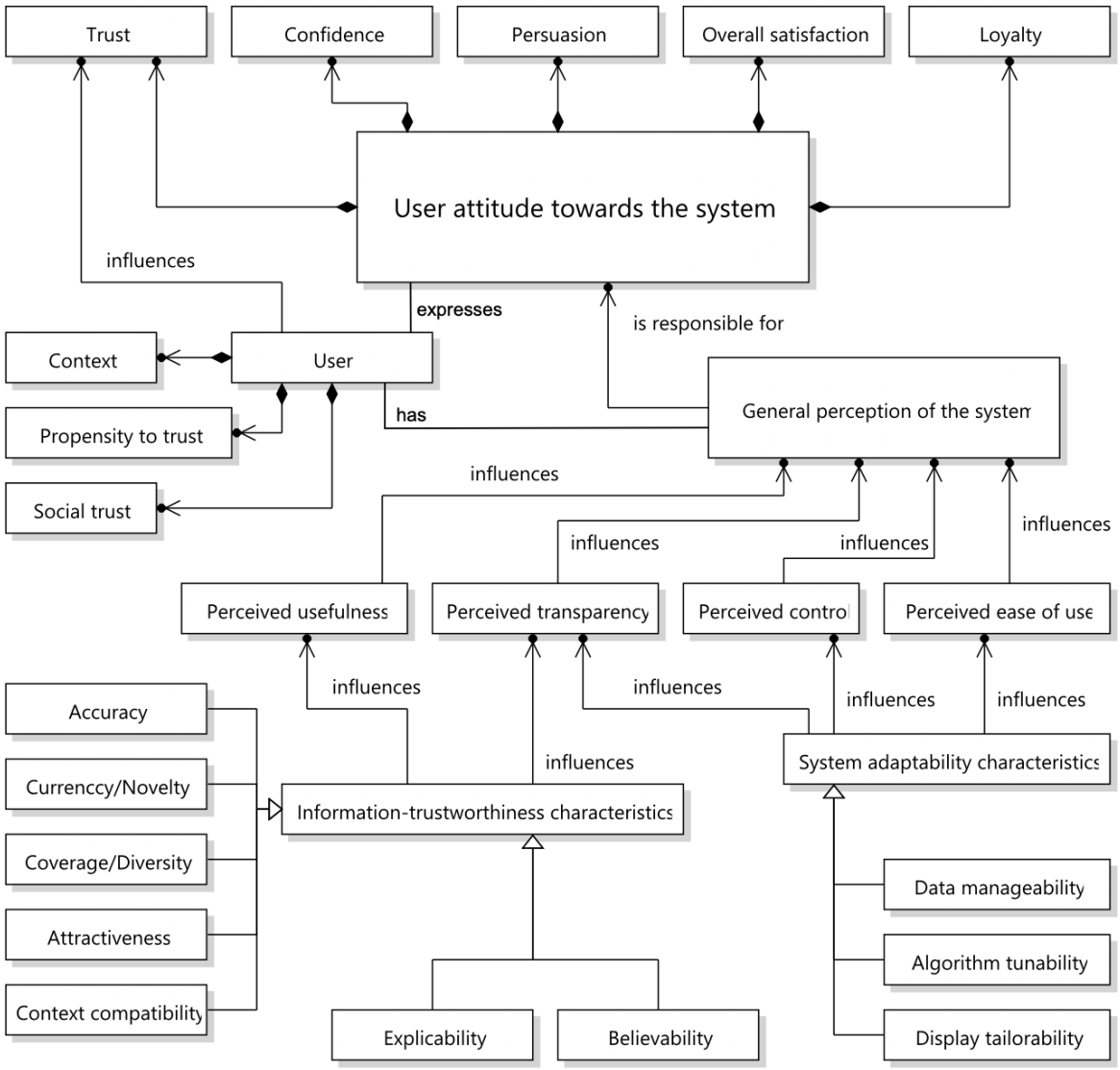



Figure 6.1 – A conceptual model of trust for recommender systems.

- **Overall satisfaction** determines what users think and feel while using a recommender system. It gives users an opportunity to express their preferences and opinions about a system in a direct way.

- **Loyalty** refers to the system's ability to make the user use it and no other one when performing a task.

When assisting creative task, one mistake is to only pursue system accuracy and performance, what is done for a lot of artificial intelligence systems, at the expense of the characteristics listed above. The very notion of precision for a modeling recommendation is complex to grasp. As there is no single way to model a system but at least several, the accuracy of a recommendation gets totally subjective, depending on the

engineer, the context, or the business domain. A good recommendation for one engineer may be wrong for another, provided that their design approach is different. Thus, in the case of creative tasks, and especially software design, accuracy or performance are criteria to take into account, but at the same level as others that influence trust, persuasion, satisfaction, and loyalty.

### Trust resulting from the general perception of the system

Figure 6.1 demonstrates the influence of *perceived usefulness* (including information quality), *perceived transparency*, *perceived control*, and *perceived ease of use* on trust. Perceived usefulness refers to the quality and relevance of the presented information. Perceived transparency represents the ease of understanding of how the system works. Perceived control captures the ability to adapt, while perceived ease of use relates to how easily a random user can start interacting with the system. Together, they influence the *General Perception of the System*, including trust. These four characteristics can, in turn, be influenced by information-trustworthiness characteristics, system adaptability characteristics, or both.

Information-trustworthiness characteristics, such as accuracy or currency, are essential to the development of an impression of perceived usefulness. Others, such as believability/explicability or attractiveness, can also strengthen the impression of perceived transparency. Our study of the literature identified the following information-trustworthiness criteria:

- **Accuracy.** The extent to which each recommendation matches the current element for which recommendations are requested.
- **Currency/Novelty.** The extent to which each proposed alternative is commonly used in similar contexts.
- **Coverage/Diversity.** The extent to which the full set of suggestions covers the different possible modeling paths, semantics of the objects, or user styles.
- **Attractiveness.** The extent to which each suggestion might be interesting for the user. It includes matching users' preferences, but also company-internal good practices, or general modeling good practices such as design patterns.
- **Explicability.** The extent to which the presence of each alternative in the list of suggestions can be explained, based on explanation

elements that resonate in the current context. It refers to the reason why the element is in the list.

- **Believability.** The extent to which the user can be convinced of the relevance of each suggestion in the list. This criterion may refer to authority figures (such as company experts), or user statistics to convince. It transcribes the reason why you might want to select this alternative.

- **Context compatibility.** The extent to which each alternative fits the current context, including the whole model and user's intents.

System characteristics relate to system features that are responsible for creating and displaying recommendations. Interfaces with explanations can influence the perceived transparency. Some algorithms might accept feedback from the user and enable user's perceived control. The metamodel presented in Figure 6.1 relies on the following characteristics of system adaptability:

- **Algorithm tunability.** This characteristic transcribes the ability of the algorithm to be adjusted by the user, to change and refine its behaviour, and the recommendations it produces.

- **Display tailorability.** The extent to which the user interface views can be customized by each user, so they can create a visual environment that fits their expectations.

- **Data manageability.** The extent to which the database of the system can be trimmed, updated, or augmented by the users.

In this thesis, we focus on the influence of perceived usefulness, perceived transparency, perceived control, and perceived ease of use towards increasing the trustworthiness potential of our system. In particular, we address the following characteristics: *information trustworthiness*, *information transparency*, *system transparency*, *system control*, and *system ease of use*.

### The impact of individual social and cultural backgrounds

Although the nature of a system, its components, and its operations naturally influence whether or not a trust relationship is established with the user, other factors can complicate the development of this relationship. The previous sections have highlighted the characteristics of the information system that can influence trust, notably through the

information presented by the system. However, in the human-system relationship, the characteristics of the human also affect the perception of trust.

In their paper, Baer et al. [13] show that the social context has a substantial impact on dyadic trust. This social context includes the *status*, defined as the prestige, respect, and admiration that individuals enjoy in the eyes of others by Lount and Pettit [100]. In their paper, Lount and Pettit observed that the possession of high status led individuals to be trusted more. Personnality traits are recognized as a factor that might influence users' propensity to trust other colleagues or systems that they interact with [39]. This is for instance the case of *openness to experience*, one of the five domains of the Big Five personality traits model [46], which appears to be positively correlated to trusting others more [6]. This propensity to trust can also vary according to the cultural background of users. In their study, Miller and Mitamura [114] observed that the notion of trust and risk was perceived differently between participants from America and Japan.

These observations from the literature complicate the generalizability of the impact of trust factors in a system on the actual trust relationship established. Depending on the target population, but also on the profile of the users, the criteria mentioned above may have a different impact, more or less important. These observations also have an impact on the evaluation process of the trust relationship. The empirical validation process of the results on trust, which is generally based on questionnaires capturing the users' feelings, must be thought out and tailored to the target population of the experiment.

### 6.1.3 Approach to design trust-fostering modeling assistants

Our assistant cannot produce trust, as it is something that emerges or not, or can be borrowed from another source of authority [117]. Thus, to address trust issues, we can only create an environment that enables the growth of the user-system trust relationship.

Our approach investigates the design of software assistants for software modeling. Thus, due to the software-focused aspect of our approach, we only consider trust characteristics that actually are addressed from the software. This implies that external factors that might influence the development of trust between users and modeling assistants are out of our scope.

Hence, we exclude user-related trust aspects such as the social or

cultural background, as presented in the previous section, from our consideration in this manuscript. As a consequence, software assistants should be designed by taking into account the assistant-oriented criteria that influence the user's general perception of the system as follows:

**Information trustworthiness.** To foster trust, we conclude that software assistants for modeling should be designed to express the *accuracy*, the *currency or novelty*, the *coverage or diversity*, the *attractiveness*, the *explicability*, the *believability*, and the *context compatibility* of information to perceive information trustworthiness.

**System adaptability.** The system should also enable *algorithm tunability*, *display tailorability*, and *data manageability* for the user to perceive system transparency and controlability.

## 6.2 Addressing creativity issues

As discussed in Section 2.3.3, the modeling task is a complex design task, for which software engineers call on their creativity to produce software systems that matches the design constraints. In this section, we clarify the *creativity* definition for software modeling, and identify architecture guidelines from the literature to build software that fosters creativity.

### 6.2.1 Framing creativity for software modeling

Creativity is one of those concepts that have led to an enormous production of papers in the literature, at some point that there is even an Encyclopedia of Creativity [150], and a *Handbook of Creativity* [171]. Today, this notion is a research topic in itself that still fuels scientific debates. However, there is a consensual definition accepted by most researchers in the field. Creativity is the ability to produce something that is both (i) new and (ii) adapted to the context in which it occurs [102]. A production that is (i) *new* is by definition original and unexpected, distinguishing itself from what the subject or others have already produced. This novelty can be minimal, differing only slightly from previous solutions, or present a significant break with previous achievements [172]. However, the creative output cannot only be new but must also be (ii)

*adapted*, satisfying the different constraints related to the context for which it is designed. There are also other characteristics that influence judgments about the creativity of a production, such as the technical quality of the solution or the importance of the solution in relation to the needs of society.

In software engineering, creativity as defined above can sometimes be complicated to detect. Indeed, the lessons learned since the beginning of computer science have led researchers and practitioners to develop ways to avoid falling into both financial and technical pitfalls. For instance, the definition of development cycles and specific methodologies such as the agile method have aimed to standardize and structure the stages of the software life cycle, in order to control the risks and minimize the chances of failure of the project. The frameworks aim at standardizing the way of designing the architecture of systems of a given type, as does this thesis for software modeling assistants. In the same way, best practice rules aim at standardizing the way of coding (naming conventions, code layout, and many others) within a community of software engineers, a company, or even a project team. All these resources aim at unifying the way of solving problems in software engineering, seem to go against the notion of novelty mentioned before, and thus against the very notion of creativity. In practice, they just add new constraints that creative solutions must match.

Designers have to adapt to time constraints, financial constraints, constraints emanating from the customer's needs, but also constraints related to the existing ecosystem in which the solution must be integrated, related to the experience of the modeling team, or related to the requirements of the business domain and its possible approval organizations [12, 178, 165]. These technical, financial, and human imperatives set up a perimeter that delimits the boundaries of the acceptable novelty of the solution. The designers of software systems must then work within this perimeter to find a solution that maximizes the respect of the constraints. It is during this maximization process that creativity emerges for the software design task and more specifically for the software modeling task. As presented in Section 2.3.3, the designer explores the problem space and then the solution space in a progressive and iterative way in order to produce his model, as synthesized in Figure 6.2. The constant evaluation of this temporary model then allows its refinement, which in turn leads to the emergence of new problems to solve. The designer's ability to be creative during this process is then essential to the discovery of an optimal architecture in the solution space of the design problem. In software design, creativity can then be assimilated to the designer's

ability to combine technical bricks, business concepts, and good architecture practices to solve these successive problems, within the perimeter delimited by the project constraints.

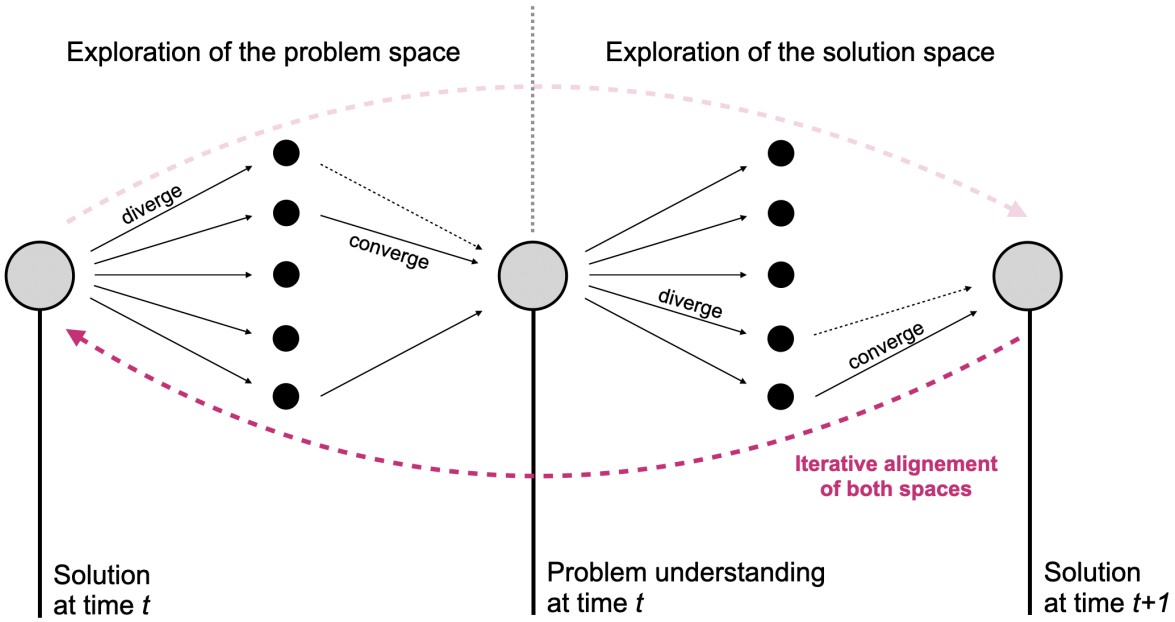



Figure 6.2 – Iterative alignment of problem and solution spaces, adapted from [99]

### 6.2.2 Understanding creativity characteristics for software support

In order to support the development of creativity through software tools, it is necessary to identify the factors that influence it. Since 1980, the idea of creativity arising from several characteristics has been called the multi-faceted approach to creativity. This approach describes creativity as dependent on cognitive, conative, affective, and environmental factors, presented in Figure 6.3. These four main factors have been identified as a result of much work in the area of understanding creativity as identified in the book by Lubart et al. [102]. According to them, each individual has a particular profile on these different factors. Thus, the creativity potential of an individual in various fields of activity results from the combination of these different factors in the social and technical context of the project.

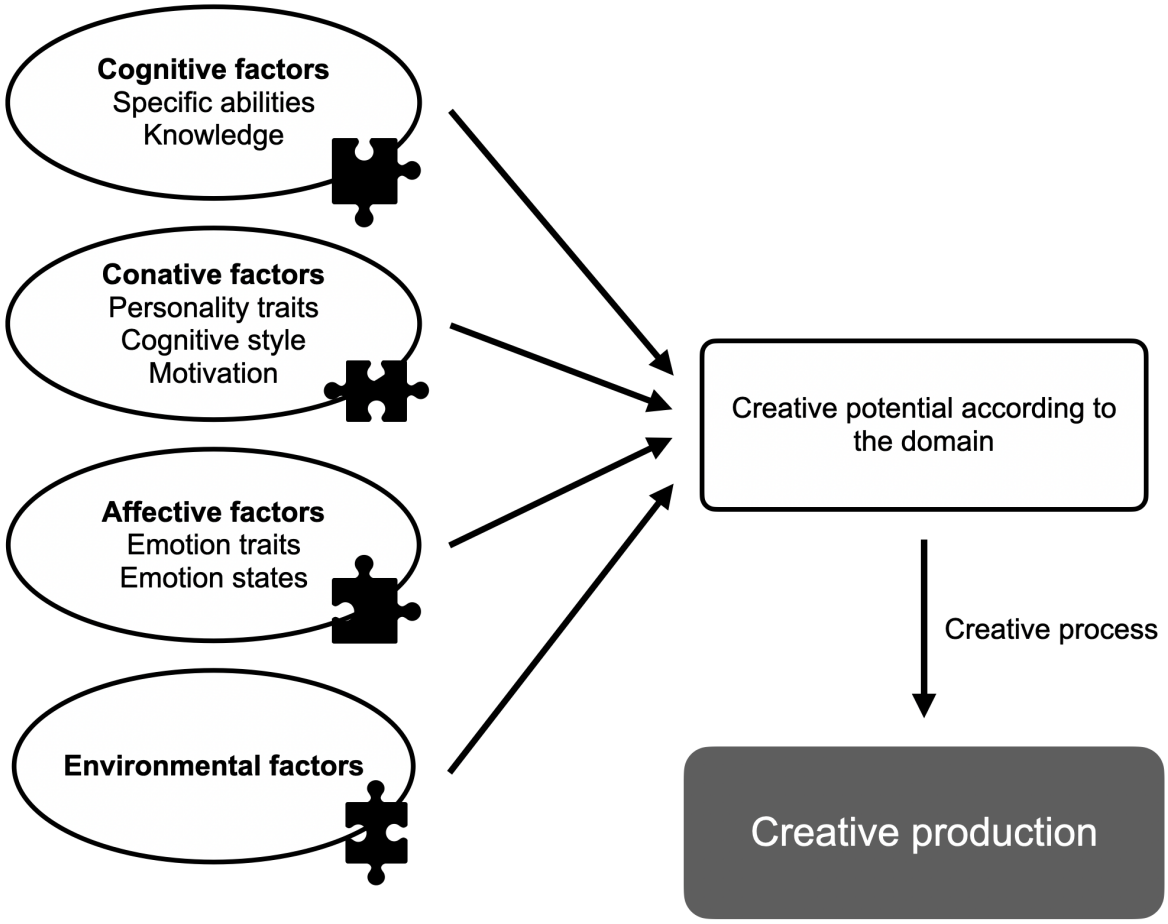



Figure 6.3 – Main factors for creative production, from [102]

### Cognitive factors

Cognitive factors include the notions of *intelligence* and *knowledge* of the software engineer. From the perspective of Batey et al. and Lubart et al. [16, 102], the intellectual abilities considered essential in the creative act are those that serve (i) to identify, define, and redefine the problem (or task), (ii) to identify information in the environment that is relevant to the problem, (iii) to observe similarities between different domains that shed light on the problem, (iv) to group together various pieces of information that, when combined, will form a new idea, (v) to generate several possibilities, (vi) to self-assess one's progress towards the problem, and (vii) to disengage from an initial idea in order to explore new paths. Knowledge, on the other hand, refers to the information stored in the software engineer's memory, which results from their education and past experience. According to many authors, creativity can only be exercised once a certain level of knowledge is reached. This knowledge allows to understand situations and not to reinvent what already exists, but also to take into account and to take

advantage of events observed by chance [102].

### Conative factors

The literature has identified three conative factors influencing creativity. First, *personality characteristics* refer to characteristics that are relatively constant over time in relation to the individual [78]. It has been shown that personality traits related to perseverance, tolerance of ambiguity, openness to new experiences, or risk-taking have a positive influence on an individual's creative potential. Next, *cognitive styles* are described as the different ways in which individuals prefer, or tend to perform their mental actions. For example, we can differentiate between the global style, where an individual focuses on the general aspects of a task, and the detailed work style, where an individual focuses on the details of the task. The study of these different styles, although very recent, has shown that the global style seems to be more conducive to creativity. Finally, the individual's *motivation* seems to have a different impact on creativity depending on its nature [102]. If intrinsic motivation, relating to the internal desires of the person satisfied by the accomplishment of the task, seems to positively influence creativity, it is not the case of extrinsic motivation. Extrinsic motivation, stimulated for example by obtaining a reward following the task, seems to have a negative effect on creativity.

### Affective factors

Different approaches have aimed at understanding the relationship between feelings and creativity in research, such as the naturalistic approach and the experimental approach. Although numerous and informative, the results of these different types of research are sometimes divergent and make it difficult to interpret them. These studies have focused on the impact of positive emotions on creativity, and have confronted the points of view of different researchers [102].

### Environmental factors

Although the study of the creative environment is relevant to the field of creativity, Lubart et al. [102] report that less than 5% of articles are referenced by the word environment and less than 1% by the keyword social environment. The notion of creative environment can refer to several systems studying the environment of the individual at different scales. Bronfenbrenner [30] defined four levels of environments called

*microsystems*, *mesosystems*, *exosystems*, and *macrosystems*. The levels respectively consider more environmental factors, from local to global. For instance, microsystems relate to the study of the subjects' evolution in individual communities to which they belong, while mesosystems consider the interactions of these various communities brought together, such as combining religious education with school. The two other systems broaden the scope even more, to embrace many more environmental variables.

In our case, we are interested in the more local environment, only related to the *work context* and the *work team* and thus focus on microsystems. These microsystems include the social groups in which the individual participates, and more particularly the team in which the individual works. This team can be made up of human collaborators, but also of machines, with which the individual weaves a working and affective relationship. Lubart et al. [102] state that the probability of innovating at work will be facilitated if the work is done in a structure that allows and encourages the creativity of each of its members. They identify that this encouragement can be achieved, for example, by offering access to databases and information at work.

**Which characteristics do software assistants for software modeling support ?**

The factors of creativity discussed above can thus all have an influence on the overall expression of an individual's creativity. Our description of the modeling task in Section 2.3.3, however, highlights different constraints on this creative design activity, such as its work-related and ill-structured nature. Moreover, the use of a modeling software assistant, intervening as an external actor different from the actual individual performing the task, limits the creative factors that can be addressed. It seems unlikely that the assistant can intervene correctly on the conative and affective factors that are mostly related to personality traits, personal cognitive styles, or feelings. On the other hand, the nature of the cognitive and environmental factors are compatible with the capabilities of the assistant.

Cognitive factors refer to the notions of intelligence and knowledge. For several years now, the notion of artificial intelligence has fully taken its place within the software engineering community, at the origin of systems aiming to carry out so-called intelligent tasks generally performed by humans. These systems are becoming increasingly commonplace and the research effort in this field has never been so sustained. However, the objective of creating intelligent systems is slightly diverging towards

a definition that leaves room for humans. The notions of *collective intelligence* [107] or *augmented intelligence* [44, 149] have made their entry into the scientific community. Instead of doing *in place* of humans, these new systems aim at doing *with* humans, helping them on increasingly complex tasks. This new paradigm of collaboration is intended to take advantage of the thinking capacity of both the human and the machine, in synergy, in order to increase the intelligence potential of the pair. Modeling assistants share this objective, to augment users by supporting their potential for confidence, instead of replacing them. This is achieved by making knowledge from various sources and domains available in an easily accessible way. Thus, the assistant might also help with the tasks related to the notion of intelligence mentioned above, such as grouping together similar pieces of information, or proposing new possibilities or avenues of work, by presenting this knowledge in forms that encourage reflection according to the individual.

In these cases, the assistant must be integrated into the user's working environment, for example by being integrated into the modeling tool. The system can then influence the general relationship that the user develops with the modeling tool, while allowing the development of an affective relationship between it and the user. This relationship can for example be expressed through the user's pleasure in using the assistant. For this, the assistant must be compatible with the users' working style, by adapting to their habits and way of working.

The identification of these characteristics of creativity allows us to define guidelines for the support of creativity within Software Assistants for Software Modeling (SASM).

### 6.2.3 Software characteristics to support creativity

We gathered several papers [74, 24, 193, 145, 167, 188] presenting frameworks or guidelines for defining software systems capable of supporting the creativity of their users. The results presented in these articles matches the study of the factors of creativity that can be influenced by software assistants presented in the previous section. In this section, we present the four main identified axes of creativity support for modeling assistants *knowledge availability*, *knowledge representation*, *supporting users' paths and styles*, and *usability and transparency*.

#### Knowledge availability

One of the system prerequisites to support creativity is the availability of knowledge. Hewett and Wang et al. [74, 188] suggest the use of a

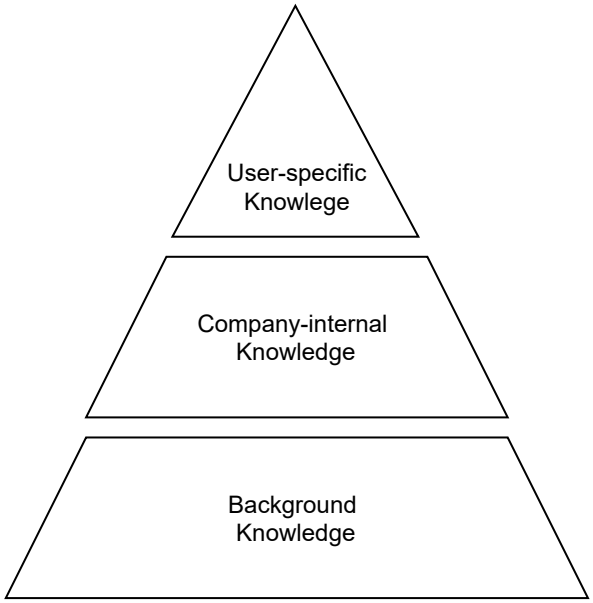



Figure 6.4 – The three levels of knowledge for modeling assistants.

knowledge base to facilitate its access. This knowledge can be varied and can come from different data sources. In the case of modeling assistants, this knowledge can take the form of models that have already been built, but it can also come from the analysis of documentation or requirements. To support the reflection and analogy process in the creative process, Bonnardel and Marmèche [24] suggest to vary the sources and domains of knowledge.

Thus, we propose that the knowledge base be built around three levels of knowledge as presented in Figure 6.4. The *background knowledge* is composed of general knowledge on various domains and common concepts. The *company-internal knowledge* allows to refine the concepts specific to the company and its field of activity. The *user-specific knowledge* is specific to users and carries the knowledge of their working habits. We formulate the following requirement for modeling assistants.

**Knowledge base.** The modeling assistants must feature a knowledge base built around three levels of *background*, *company-internal*, and *user-specific* knowledge.

As specified by Resnick et al. [145], the knowledge base must be large enough to allow a real exploration of the solution domain in the business domain but also elsewhere. The size of this exploration space must be communicated to the user for the sake of transparency.

In order to feed this database, the system must be able to collect data during its operation. Hewett [74] mentions in particular the process

of acquiring feedback from the user. This collection must be simple, integrated into the work process and non-disruptive. In the same way, the authors mention the need to be able to access, archive, and analyze the communications of software engineers in order to extract the contextual information necessary for the acquisition of knowledge about the project. The models produced by the users, but also their team, the company, or a community of developers must be recovered and progressively integrated into the knowledge base. These different techniques make it possible to feed the base on the three levels of knowledge to keep it up to date.

> **Knowledge availability.** The modeling assistant must set up a system to guarantee the availability of knowledge. This may be achieved through the retrieval of edited models, but also from the analysis of work artifacts, and professional communications.

### Knowledge representation

Beyond making knowledge available, the modeling assistant must help the user to accomplish cognitive tasks related to creativity. These tasks rely on the ability of the software engineers to put their personal knowledge and external knowledge into perspective, in order to navigate through the different possible solutions. In this case, the assistant has a determining role in the way of presenting the knowledge to the user, through its interface. Wang et al. [188] insist on the importance of the choice of the data representation according to the task to be supported.

> **Knowledge representation choice.** The choice of the knowledge representation in the user interface of the modeling assistant must be made according to the software engineering task that is aimed to be supported.

Three of the considered papers [74, 193, 188] highlight the fact that in order to support creativity, a software system should ideally be able to propose an interface allowing to change the point of view on the knowledge.

> **Knowledge representation switch.** The interface of the assistant must allow the user to navigate between several representations of knowledge, in different forms.

These same three articles mention the problem of accessing and manipulating these representations during creative work. In a physical environment, working with a whiteboard, pens, markers, and sheets of paper, for example, enables people to arrange their workspace as they wish. For instance, software engineers can place several sheets of paper next to each other on their desk to visualize different documents, while exploiting on the whiteboard, where they can easily write, erase, or trace elements. The immediacy of access to different representations, as well as the ease of arranging the workspace, are problems that can be transposed to the software assistant. Thus, we formulate the following requirement.

**Ease of knowledge access.** The modeling assistant should allow a fast and easy access to the knowledge, and should allow a simultaneous display of several knowledge representations.

### Supporting users' paths and styles

Rather than supporting the conative factors of creativity, the assistant can instead adapt to them to enable a much more tailored user experience. Each individual has their own way of working, such as starting from the global to the detailed, or vice versa. Some designers will prefer to sketch the whole model before detailing it, and others will take care to specify each model element before moving on to another. These habits, which are unique to each individual, can quickly be undermined if the system (i) does not allow for customization or (ii) imposes too many constraints on the user.

In his paper on the design of computer-based environments to support creativity, Hewett [74] evokes the notion of *tailorable environment*. In order to adapt to the user, the assistant must not be designed as a monolithic block, impossible to adjust. This notion of adjustment refers to the graphical aspect (placement of the assistant views), to the access to its functionalities (keyboard shortcuts, menu entries), but also to its internal functioning. Users should be able to adjust the behavior of the system according to their needs and to their liking, for example to receive more or less recommendations, or to ignore certain elements if they are sure of themselves.

**Tailorable environment.** The modeling assistant should be adjustable according to the user's preferences and needs on the aspects of triggering, display, and internal behaviour.

At the same time, in addition to being configurable, the assistant must allow the user freedom. As mentioned in the previous section, the solution space for the creative problem is limited by many project constraints. The software engineer must then find a solution within this space, even if it sometimes means to think beyond, to *think outside the box* related to the notion of *functional fixedness*. The software system must allow the user to carry out these problem/solution iterations, without technical or syntactic constraints, during this stage of the modeling process [74, 145].

**User freedom.** The assistant should allow users to conduct their creative process as they like, and should not impose any constraints on the travels in and out the solution space.

### Ensuring usability

Resnick et al. [145] refer to the notion of accessibility with the expression *low threshold, high ceiling*. This first implies that the assistant must allow any beginner to use it in a simple way to accomplish tasks (low threshold). This accessibility is emphasized by Hewett [74], who suggests to provide a user manual for the assistant in a systematic way to the user. At the same time, the assistant should not limit users and allow them to perform more complex tasks if they wish (high ceiling). However, Resnick et al. recommend keeping the assistant simple *and even simpler*. In other words, the assistant should only be able to perform a limited number of tasks. Supporting more tasks would then require to define several assistants, that could be installed according to the need. We formulate the following requirement.

**Low threshold, high ceiling.** The modeling assistant should be accessible to beginners, while allowing experts to perform their modeling tasks. The number of features of the assistant should be restricted to the very minimum, matching the requirements of one modeling task.

Beyond its accessibility, the assistant must be transparent about its

scope of action and its interoperability. For example, it must indicate whether the knowledge can be used in other tools, but also whether the system can allow modeling for other domains, for other types of diagrams.

**System interoperability.** The modeling assistant should support interoperability, by being compatible with different modeling tools or different business domains.

## 6.3 Building a Recommender System for modeling

Recommender Systems aim to reduce information overload by retrieving the most relevant information and services from a huge amount of data [101]. Their main feature lies in their ability to adapt to the user and the context to generate personalized recommendations. Modeling assistants are expected to manage a huge quantity of knowledge, from different domains and scopes. Then, they propose new features exploiting this knowledge to their users. A fundamental and unavoidable component of software assistants for software modeling is therefore a recommendation system. Whether they are fully autonomous, or they let users be part of the decision process, software assistants rely on recommender systems that enable them to handle the concept of choice and alternative selection. However, all recommendation approaches might not fit the requirements of recommending for modeling. In this section, we discuss how multi-criteria recommender systems appear as the most suitable systems for supporting modeling, and then we elaborate on how to design them based on frameworks from the literature.

### 6.3.1 Single-criterion recommendation approaches for modeling

Most recommender systems implement one of the following three common recommendation methods: collaborative filtering [160], content-based techniques [132], or knowledge-based techniques [36].

Collaborative filtering methods rely on a database of rating from various users. It is based on the assumption that if two people agreed in the past, they are likely to agree in the future. Then, it finds people sharing most of the current user's opinions (based on previous ratings),

and use people ratings about a concept X to predict the current user's opinion about X.

In content-based techniques, the system tries to identify common characteristics between the elements that the user liked, and exploit these characteristics to predict new elements that might match the user's expectations.

Knowledge-based recommender systems exploit explicit information from the user and item characteristics to identify elements of interest. This kind of approach allows users to define constraints on the expected suggestions, which are exploited to produce the recommendations.

Each one of the previous approaches has advantages and limitations, which include overspecialization, cold-start issues, and scalability issues [101]. Beyond these technical issues, these recommendation methods either rely on (i) the assumption that the recommendation is a matter of taste or opinion, (ii) the possibility to identify common characteristics between alternatives, or (iii) the users' ability to identify and define constraints on the recommendations they expect. While these are an option for different domains such as online shopping, movies, video games, or real estate recommendations, this is an issue for the application of these recommendation approaches to modeling. Firstly, (i) a design decision or the choice of the elements of a diagram is hardly or not at all a matter of personal taste of the engineer but is governed by budgetary constraints, be they financial, human or technical. Secondly, (ii) the making of such decisions depends on the context, the scope of the diagram, the recipients of the documents produced, or the criticality of the system to be implemented. Thus, during the same work session, the engineer may exhibit antithetical behaviors when producing two different artifacts, in the same work environment. It is then impossible to identify characteristics common to the choices made by the user. Finally (iii) the problem-solving aspect of the modeling task implies that users do not necessarily have an idea on how to solve a problem, and are therefore not able to define what they expect from the recommendation system. To solve these problems, different advanced recommendation approaches have been proposed including Multi-Criteria Rating Recommender Systems (MCRS), which allow to consider several aspects of a context to produce suggestions [108].

### 6.3.2 Multi-Criteria Rating Recommender Systems

Multi-Criteria Rating Recommender Systems (MCRS) are systems that involve multiple criteria in selecting the final set of ranked recommendations. Adomavicius and Kwon [4] identify multiple selection techniques,

such as rating-based, multi-objective optimization, or outranking relations. To fully support creativity, no single recommendation approach is sufficient. This means that, in the context of software modeling, the generation of a recommendation set cannot be achieved by means of simple logic. Instead, it must be based on the notion of proximity to an objective. Because of this and because the quality of a recommendation depends on more than one measurable criterion, we chose to build our system as an MCRS.

In their systematic review of MCRS, Adomavicius and Kwon [4] apply Roy's methodology for analysing *multi-criteria decision-making* problems [148] to MCRS design:

1. **Defining the object of decision.** The system scope needs to be delimited first. This is done by identifying the potential solutions (i.e., alternatives).

2. **Defining a consistent family of criteria.** A set of functions (i.e., criteria) must then be identified and described in order to assess each alternative over the different parameters affecting the choice of recommendation.

3. **Developing a global preference model.** The role of the global preference model is to combine all criteria into a single model that is used to make the final selection from among candidate alternatives.

In defining our approach, we applied the above methodology to our MBSE problem.

### 6.3.3 Defining the object of decision

Determining the object of decision is the way to define the scope of our system. This is necessary to more precisely define its inner mechanisms. In our approach, the object of decision consists of determining all context elements that might be involved in deciding whether a candidate alternative represents an appropriate recommendation. In the context of MBSE, we propose to define the **object of decision** as the combination of the following *characteristics of a recommendation*: its *Nature*, its *Model Scope*, its *View Scope*, and its *Target*. To support the understanding of these different concepts, let us assume that for example, we aim to design a recommender system that suggests class attributes in UML class diagrams.

The Nature of a recommendation refers to the information it conveys, that is, the modeling language meta-type of the recommended element.

In our example, this is the UML meta-type *Attribute*. For the Model Scope of a recommendation, we define the set of elements from the modeling language that can own, in their containment hierarchy, elements of the recommendation *Nature*. In our example, the Model Scope of the system consists of the following meta-types set: *{Class, Package}*, both of which can own attributes in their hierarchy. View Scope represents the top-level view in which the elements of the recommendation Nature are represented. For our example, the View Scope is a *Class Diagram*. Finally, Target designates the user for whom the recommendations are intended and personalized. Users of our example system are expected to be *users of modeling tools, possibly with some previous experience, their own personal preferences, and with individual degrees of willingness to extend trust in given situations*. The latter psychological trait is referred to as the *propensity to trust* [41].

Adomavicius and Kwon [4] require that the definition of the object of decision should indicate how recommendation alternatives are to be selected through a *decision process*. This is done by means of a *description*, which specifies each alternative in terms of how it performs for each of the multiple criteria. This procedure ensures that only elements that perform well on one or more criteria will be considered as suitable candidates. This provides the first mechanism for exclusion of elements from the recommended set of alternatives. *Ranking*, *sorting*, and *choice* approaches [4] were discarded because they needlessly consider all the elements, which is unrelated to semantics and which can also cause performance issues. Instead, potential recommendation candidates are determined solely according to the description decision process. Consequently, we define the object of the decision of our modeling assistants as follows:

> **Object of Decision** Elements (i) selected by their *Nature* or their presence in the hierarchy of elements from the *Model Scope*, represented in the *View Scope*, and tailored to the *Target* users.

We can apply this definition to our example system. Then, we describe the object of decision of our example as *attributes (i) selected by their nature or their presence in the hierarchy of classes or packages, (ii) represented in class diagrams, and (iii) tailored to the user of the modeling tool, who may have previous experience, personal preferences, and various levels of propensity to trust.*

### 6.3.4 Criteria Identification

Our analysis of several publications identified that the confidence potential of a recommender system was a function of the general perception that the user has of the system (see 6.1.2). More specifically, the perception related to information and the perception related to the interface and interactions both appear to be correlated with the notion of trust in recommender systems. Because they characterize information, identification criteria depict the information-related perception of the system. In this section, we focus on how to identify a family of criteria that promotes the perception of **information trustworthiness** and **information transparency**.

The transparency of a process, situation, or statement is defined in the Collins Dictionary as its *"quality of being easily understood or recognized, for example [. . . ] because it is expressed in a clear way"*. Information transparency would be achieved by making the inner mechanisms of our system easily understandable to end-users. Therefore, information transparency calls for explanation. Because of the multi-criteria nature of our system, transparency can be realized by means of a two-step explanation: users need to first understand what each criterion individually stands for, and then how the different criteria are aggregated into a final ranking. While this last step will be addressed in Section 6.3.5, the first step requires the criterion definition to cover explainability. We propose that each criterion is described by a concise rationale, which includes only the amount of technical detail tailored to the level of technical expertise of the end user. Thus, the first semantic constraint on criterion definition is:

**Criterion Constraint 1: explainability**
A criterion shall be described through a rationale that is easily understandable by the recommendation target.

Section 6.3.2 introduced the six information trustworthiness characteristics for recommender systems: *accuracy*, *currency/novelty*, *coverage/diversity*, *believability/explicability*, *context compatibility*, and *attractiveness*. These factors, also referred to as *information quality* characteristics, are qualities that information must reflect in order to improve the general perception of users about the system. Expressing them through criteria would positively influence the relationship of trust of users towards the system. Therefore, the second constraint on criterion definition is:

**Criterion Constraint 2: trustworthiness**
A criterion shall reflect at least one of the following information trustworthiness characteristics: accuracy, currency/novelty, coverage/diversity, believability/explicability, context compatibility, and attractiveness.

The suitability of a recommendation is clearly dependent on its context. For UML attributes, the class that is being described, the content of the class diagram, or the profile of the user, are all different elements that must be taken into account to provide coherent and meaningful recommendations. Such context elements are accurately identified in the previous definition of the object of decision of our system. Consequently, the third constraint on criterion definition is:

**Criterion Constraint 3: context**
A criterion shall exploit information from at least one of the four components of the object of decision. This relates to the *nature* of the recommended alternatives, their presence in the hierarchy of elements from the *Model Scope*, represented in the *View Scope*, and tailored to the *Target users*.

### 6.3.5 Utility Function

In defining what makes a good explanation in recommender systems, Tintarev and Masthoff [182] argue that *"justifying [a] recommendation is just half of the solution, the second half is to make it scrutable"*. To that end, in this section we first select an aggregation method that enhances system transparency. Then we emphasize support for context adaptability, and, finally, propose a determination process that allows system control through scrutability.

#### Utility Function selection

Adomavicius and Kwon [4] identify two major techniques for dealing with multi-criteria ratings to produce an overall rating: heuristic-based and model-based techniques. Heuristic-based techniques compute the score of each item for a given user, based on data derived from observing that user, using some heuristic assumption. To perform matching operations, these techniques often require specific knowledge about multiple users, based on their profile and from collaborative filtering. In contrast,

model-based techniques generate a predictive model, typically using statistical or machine-learning methods that best explain the observed data. Once the model becomes available, they use it to estimate the score of individual recommendations.

In the case of software assistants, the lack of data about the profiles of all users rules out heuristic-based techniques. On the other hand, model-based techniques using machine-learning methods enable the system to learn directly from the user, resulting in finely-tuned data. Consequently, we take a machine-learning model-based approach to determine the overall utility function. Note that, for greater system transparency, the aggregation process must be explainable. Therefore, rather than relying exclusively on machine-learning processes, which are rarely fully explainable[2], we define the utility function as a weighted sum of criteria rating functions.

We define this function as follows:

**Aggregation function.**

Let $(w_1, w_2, ..., w_n) \in [0;1]^n$ where $\sum_n w_n = 1$,

$$\begin{array}{rccc} overall & : & \mathcal{A} & \rightarrow & [0;1] \\ & & a & \mapsto & \sum_n w_n \times s_n \end{array}$$

with $s_1, s_2, ..., s_n$ the individual scores for the $n$ criteria
$w_1, w_2, ..., w_n$ the weights for the $n$ individual scores
$\mathcal{A}$ the set of recommender alternatives

(6.1)

The weights $w_1$, $w_2$, ..., $w_n$ can either be manually defined by the designer of the assistant, by the user who can experiment with different configurations, or determined through a machine learning process.

### Context adaptability

Adomavicius and Kwon [5] also note that the aggregation function can have different scopes: total (i.e., when a single aggregation function is learned based on the entire data set), user-based, or item-based (i.e., when a separate aggregation function is learned for each user or item). Thus, it is possible to define different contexts in case when only a subset

[2] A dedicated research community is very active in exploring this topic. Thus, this may change in the future.

of the criteria is useful to compute the overall score. For instance, the overall score for recommending alternatives in a model containing only one element might take less criteria into account than recommending for a model with dozens of elements, which is more complete and precise.

Consequently, we refine the overall utility function, by taking into account the context $k$. We define the function $overall_k$ as:

**Context adaptability for aggregation functions.**

$$overall_k(a) = \sum_n w_{n,k} \times s_n \tag{6.2}$$

Then, there are as many *overall* functions as there are different contexts. The weights for each function must be defined separately, to ensure the consistency of the calculation in each of the different contexts.

### Utility function determination

The quality of a recommender system depends primarily on its ability to propose items that the user is likely to choose rather than items the user is unlikely to choose. Therefore, a high-quality recommender system must fit user preferences. Our system offers the possibility to reflect these preferences by assigning values to the weights of the four overall utility functions. This can be done manually, but finding suitable values would likely lead to suboptimal results. Instead, assistant designers may choose a machine-learning approach to automatically determine these weights.

Supervised learning is a common technique for inferring user preferences. Based on labelled data created by the user, supervised learning enables the system to adapt its inner mechanisms to reflect a user's decision rationale as closely as possible but without expressing it explicitly. To gather labelled data for the modeling assistant, the following scenario can be conducted. A list of unranked candidates —potential recommendations— is displayed for each situation. Using this interface, the user is asked to remove all attributes that do not fit semantically in the presented context. Once this is completed, the user is then asked to create a ranked list of the top 10 best recommendations from the displayed elements. This task should be repeated for multiple situations in different contexts a sufficient number of times in order to collect enough information to determine the four utility functions. Once this data is collected, it is used to calculate the weights in such a way that they

maximize the recommender evaluation metrics, that must be selected and computed.

Instead of gathering labelled data before the first use of the system, the assistant can be set up to collect labelled data on the fly, from monitoring the real-time usage of the system. This would imply a *cold start* phase, when the system exploits generic weights, before they are refined at the point when the user interacted enough with the assistant.

## 6.4 Designing automation

Let us assume that you join a new IT company, and integrate a new software development team. On your first day at the office, you will probably want to know how the team works, and how you will fit in the team workflow. For instance, you might want to clarify what tasks are in your scope, what you are responsible for, what resources you can get from your colleagues, but also the way you collaborate with your colleagues. Thus, from your perspective, you know what is on your grounds, and what takes place elsewhere, done by someone else. From this point of view, anything that you do not do yourself is automated, achieved by another system –your colleagues–, which might share resources with you, expect inputs from you, and produce outputs for you. This distribution within the team is the result of decades of learning and observing work teams. Optimizing the composition of teams is the role of managers who take into account the experience, skills but also the personality of employees to create successful teams.

As software assistants aim to act like a *peer* to software engineers, all the previous concerns apply to the design of the human-assistant collaboration. This set of concerns relate to the *automation* of the system, as it describes the way it is triggered, the way the user can hand-over the system, the way it gathers information from the user or the environment. This automation can be assessed through different scales available in the literature [185]. Our mapping study results introduced in Chapter 3 especially exploits the 4-aspects 10-points automation scale from the work of Parasuraman [130]. In Chapter 4, we highlighted the need for thorough automation design due to the observed inconsistencies between the available implemented systems and software modelers expectations. Thus, we base the automation design of the assistant on the dedicated framework A-RCRAFT [28, 129] which is, to the extent of our knowledge, the only available user-centered automation design framework.

The A-RCRAFT framework is a Generic Framework for Automation

Analysis and Design. It serves as guidelines to conceive a system whose goal is clear to the user, and which integrates well with the user's workflow. It provides support for the analysis of automation design over the five following aspects of automation that have to be identified at design time.

**Allocation of functions and tasks.** When conceiving the software system, the designer should clearly define which functions will be allocated to the system and which tasks will be allocated to the user.

This task distribution should be made clear to the users before they access the system, so they have a clear idea of how the system will integrate their workflow. This enables users to have clear vision of their task scope, and build expectations on the behaviour of the system in terms of what it will do, and what it will not.

**Allocation of authority.** This covers the identification of which entity is allowed to trigger or prevent functions/tasks execution.

Users of the software system should know how the system works, and how they can stop it, if they can. More precisely, users should know in advance what starting the system will trigger, and the actions they will be able to perform to pause or stop the execution of the system. For instance, if the system requires a long analysis when starting, which cannot be interrupted, and which freezes the environment, this should be announced beforehand.

**Allocation of responsibility.** This refers to understanding which entity is responsible for the outcome of the execution of the functions/tasks.

This aspect especially aims to identify who will be held responsible of any undesired outcomes of the system in case of incident. In the case of modeling assistants, it is likely that the responsibility will be allocated to the user of the assistant, e.g., the software modeler.

**Allocation of resources.** This refers to identifying which resource (e.g., diagrams, preferences, documents) is allocated to which entity in terms of production, modification or sharing with the other entity.

Users must know the consequences of integrating a modeling assistant into their working environment to evaluate the potential risks. It should be made clear whether the system had read-only permissions, or can write and edit artifacts in the modeling environment. In the meantime, the system should indicate all its information and knowledge sources, to prevent any privacy breach.

**Handover and takeover.**
When designing the system, control transitions sequences must be defined clearly, as well as which entity can trigger them.

Users of the software assistant should know how the system has to be triggered and how the system will let them take back the control of the environment. According to results of our interviews in Chapter 4, the system should be able to manifest itself, offering a non-disruptive interaction such as a notification or annotation. However, whether this behaviour can be adapted or not, whether the trigger is manual or automatic, the information should be clearly communicated to the user, and designed to offer the less disruptive interaction possible. In the case of software assistants, it is unlikely that the system will gain full handover the system, as the interaction between the user and the assistant should not disrupt the user's workflow.

## Formalizing these design constraints

This chapter identified the major design constraints for software modeling assistants, related to trust, creativity, recommender systems, and automation. These apply to modeling assistants due to the creative and problem-solving nature of the modeling task, and provide a first effort to frame their design. In the following chapter, we combine these constraints and define a formal framework for designing software assistants for software modeling.

Chapter 7

# A framework for designing SASM

*Based on the big picture of software modeling assistants that we draw in Chapter 5 and the design constraints identified in Chapter 6, we propose a general framework for designing Software Assistants for Software Modeling (SASM). This framework builds on all the work presented so far in this thesis manuscript. It aims to turn software assistants from potential solutions to concrete solutions to the modeling problems discussed in the previous chapters. Modeling assistants as presented in this chapter are hence a concrete solution to modeling issues as identified by the practitioners and in the literature.*

*We built our SASM design framework following the ISO/IEC/IEEE 42010 standard [80], which defines the notion of architecture framework and describe its content. This chapter covers the different steps of the architecture construction as imposed in the standard. It features guidelines for requirement elicitation, and provide structural and functional models to define the architecture of new assistants.*

## 7.1 SASM framework definition

At this stage of our approach, we have identified software assistants as a coherent potential solutions to many problems in software modeling. In order for such solutions to truly address these problems, it must echo the four key notions of modeling assistance. To this end, the

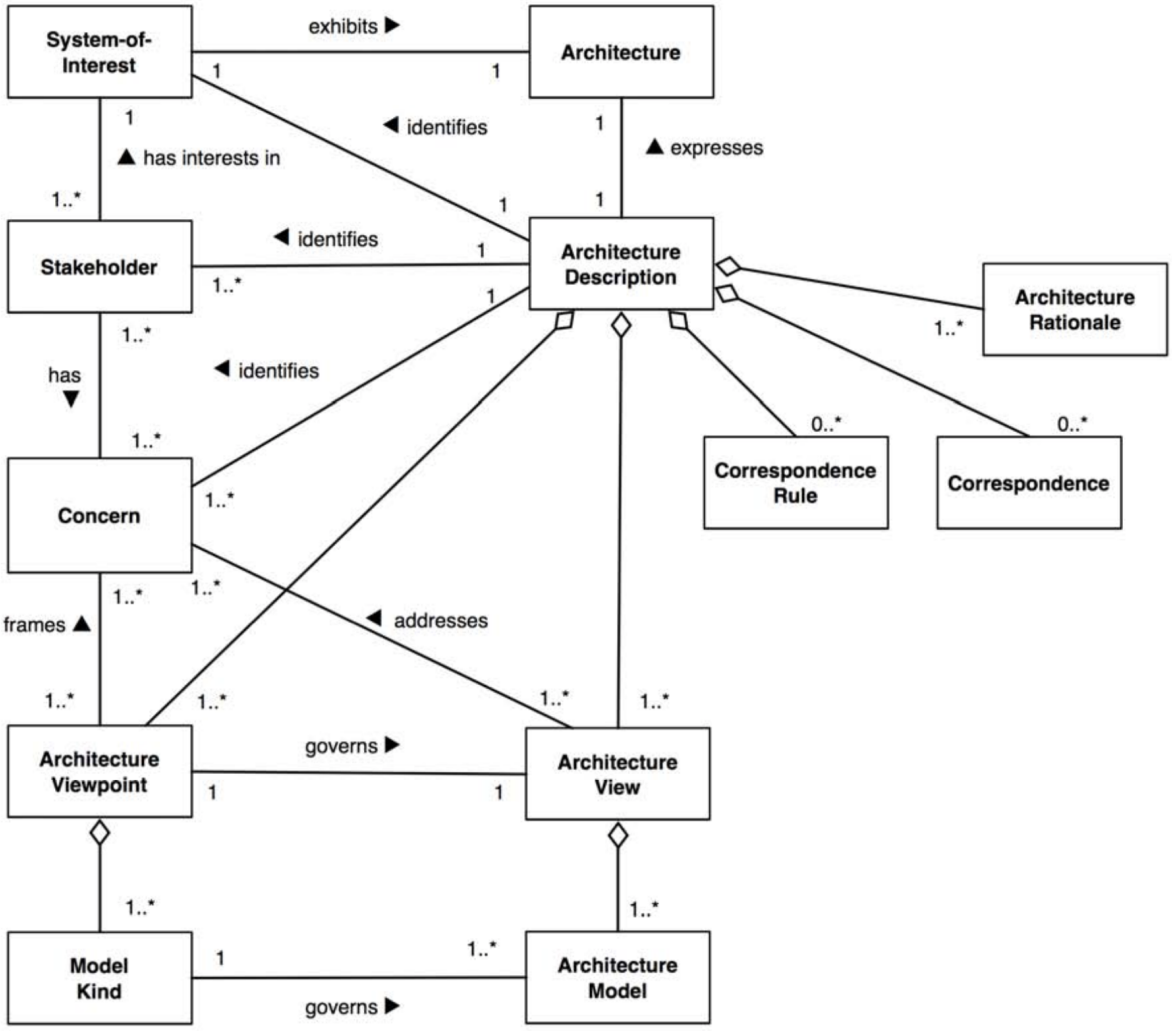



Figure 7.1 – Conceptual model of an architecture description from ISO 42010:2011 [80]

previous chapter established a set of design constraints obtained from the literature that translate these key notions. The objective of this framework is then to link all these individual constraints into a unique document that structures and guides the design of software assistants for software modeling.

The ISO/IEC/IEEE 42010:2011 standard on architecture description for systems and software engineering introduces the several components required to define an architecture framework, as presented in Figure 7.1. Thus, to define our SASM framework, we introduce (i) the architecture rationale for the system, (ii) the system stakeholders, (iii) the system concerns, (iv) the association between concerns and stakeholders (v) the architecture viewpoints, and (vi) the correspondence rules of the architecture description.

The *architecture rationale* presents the overall approach for the design

of modeling assistants. It summarizes the main characteristics that the assistants designed with the framework express.

The *system stakeholders* section introduces the actors involved in the life cycle of the assistant to be developed. As the framework should communicate information useful to each stakeholder, identifying them is the prerequisite to determining the design guidelines that the framework should include.

Based on the established list of stakeholders, the ISO standard requires *system concerns* to be listed in the framework. These concerns express the questions that each stakeholder might have about the assistant to be designed. Such questions have then be answered in the rest of the framework definition. These concerns are associated to their corresponding stakeholders in a dedicated section.

Finally, we introduce the *architecture views* that answer the previous system concerns. In this section, each concern is echoed in one or more architecture view, that describe how to design the system from a functional, a structural, an infrastructure, or a system requirements viewpoint. These architecture views are the core definition of how to design software assistants for software modeling. In a dedicated section, we link each concern to its corresponding architecture views.

## 7.2 Architecture rationale

This framework allows to design *software assistants for software modeling* defined as (i) software bots augmented with knowledge, (ii) aiming at cognitively assisting on one or several specific tasks (iii) in the context of software modeling.

Due to the nature of the task to be assisted, our framework proposes to design software assistants able to support the notions of creativity by reinforcing the trust relationship between human and assistant. The system design is based on a 3-tier architecture composed of an information repository, a knowledge back-end, and one or more user interfaces acting as clients of the back-end. The *information repository* relies on a knowledge base, built from the agglomeration of three levels of (i) general, (ii) company specific, and (iii) user specific knowledge, collected via a dedicated system. The *knowledge back-end* is implemented around a multi-criteria recommendation system generating suggestions, allowing the system to propose or make choices in an autonomous or semi-autonomous way. Finally, the one or several *user interfaces* must be integrated as well as possible with the development tools and must respect the design constraints favoring the development and the ex-

pression of trust, which is also built in the first two components of the system.

## 7.3 System stakeholders

In this section, we identify the different stakeholders of the system, according to their role, as suggested in the ISO 42010 standard. Thus, we present the different groups of stakeholders and highlight their roles in Table 7.1 Note that, in some companies, some employees might qualify for several roles in this list. Our system being based on the exploitation of knowledge, each of the stakeholders of the system is able to maintain the system, and add or update knowledge, whether technical, functional, or on the internal processes of the company.

**Modeling engineers.** They are the main users of the system, they are the ones that the assistant aims to help during modeling tasks. They are in charge of modeling the system in a general way, on all its aspects, functional and technical.

**Functional analysts.** They are in charge of the functional aspects of the solution to be designed. They know the client's needs and transcribe them into software requirements that can be presented as models. They might also analyze the design models to check the consistency of the solution according to the client's needs.

**Domain experts.** They hold the precise knowledge of the business domain concepts, and can answer questions from other stakeholders on these aspects. They are particularly involved in checking the validity of the relationships between the multiple business elements of the model.

**Client.** They are at the origin of the need to create the solution. They are the ones who fund the project and carry the knowledge related to this need, which must then be expressed as a requirement to frame the solution to be created. Thus, they can potentially help the assistant by clarifying their need, to help it better capture the intentions of the users.

**Company head.** These are the people in charge of the team that has to produce the software solution. They are concerned with the potential benefits of using the software assistant, but also the potential negative impacts of adding such a system to their employees' work environment.

**Toolsmiths.** They are the designers of the assistant, in charge of developing and maintaining the system.

| Stakeholders | Users | Owners | Developers | Maintainers |
|---|---|---|---|---|
| Modeling engineers | x | | | x |
| Functional analysts | x | | | x |
| Domain experts | | | | x |
| Company head | | x | | x |
| Toolsmiths | x | | x | x |
| Client | | | | x |

Table 7.1 – Role of stakeholders

## 7.4 System concerns

As requested by the ISO standard, the framework should identify the systems concerns, which are grouped under 5 categories (i) the purposes of the system, (ii) the suitability of the architecture for achieving the system's purposes, (iii) the feasibility of constructing and deploying the system, (iv) the potential risks and impacts of the system on its stakeholders throughout its life cycle, and (v) the maintainability and evolvability of the system.

**The purposes of the system**

This is related to the nature of the system. Software assistants for software modeling aim at supporting modeling. This can be done through 5 aspects (as identified in Chapter 4) in supporting domain knowledge, corporate methodology, modeling notation, tool usage, or modeling know-how.

*S.C. 1: How does the system assist modeling?*

**The suitability of the architecture for achieving the system's purposes.**

Chapter 6 identified the constraints to match for the system to be able to provide modeling assistance. Concerns related to the suitability of the architecture for supporting modeling are then related to trust, creativity, multi-criteria recommender systems, as criteria of a good assistants listed in Chapter 4 (usability, skills, added value, user adaptability, and context understanding).

*S.C. 2: How does the system build trust with users?*

*S.C. 2.1: How does the system enable user control?*

*S.C. 2.2: How does the system handles information-trustworthiness characteristics?*

*S.C. 3: How does the system support creativity?*

*S.C. 3.1: How does the system ensure knowledge availability?*

*S.C. 3.2: How does the system generate knowledge representations?*

*S.C. 3.3: How does the system adapt to users' paths and styles?*

*S.C. 3.4: How does the system achieve usability, transparency, and interoperability?*

*S.C. 4: How does the system implement a multi-criteria recommender system?*

*S.C. 5: How does the system respect the* good assistant *requirements?*

*S.C. 5.1: How does the system demonstrate usability?*

*S.C. 5.2: What skills does the system have?*

*S.C. 5.3: What is the added value of the system?*

*S.C. 5.4: How does the system adapt to the user?*

*S.C. 5.5: How does the system adapt to the context?*

**The feasibility of constructing and deploying the system**

This section relates to the technical aspects of the solution, including how to make knowledge available, but also the required infrastructure to deploy the modeling assistant, and to integrate the assistant within the modeling tool.

*S.C. 6: How does the system ensure data accessibility?*

*S.C. 7: What is the required infrastructure to operate the system?*

*S.C. 8: How can the system be integrated in a modeling tool?*

**The potential risks and impacts of the system to its stakeholders throughout its life cycle**

This section identifies the concerns related to the influence of the system on the modeling time and effort, as well as the potential impact of the learning on user learning and users' workflow.

*S.C. 9: How does the system aim to reduce the modeling time?*

*S.C. 10: How does the system aim to reduce the modeling effort?*

*S.C. 11: How does the system integrates to the user's modeling workflow?*

*S.C. 12: How does the system can help users learn?*

**Maintainability and evolvability of the system**

As the assistant is a knowledge-based system, major concerns are related to the management of the system when deployed in production. Different stakeholders should then be responsible for curating the knowledge, so it remains up-to-date with accurate concepts.

*S.C. 13: Can new knowledge be integrated to the system at any time?*

*S.C. 14: Can knowledge be updated in the system at any time?*

*S.C. 15: Can system knowledge be retrieved from new/multiple software clients?*

## 7.5 Concerns/stakeholders association

In this section, we provide the association rules between the concerns and the stakeholders, as presented in Table 7.2. We detail this distribution according to each group of stakeholders.

**End-users.** This includes software modelers, domain experts, and functional analysts. Their concerns relate to how the assistant might help them in their daily tasks, and on how it might empower them to save time and effort. They also care about how the system integrates their workflow, and if it requires a lot of work to be operated. They expect to use a *good* system (see Chapter 4), that might teach them new insights.

**Client.** Clients care about the quality of the produced solution, as well as the overall cost of the production. They might be involved in using the system to define or refine their needs and expectations.

**Company head.** They can show interest in providing assistance to their employees if the impact on them is positive, i.e., if it saves them time or effort, and increases their productivity. The improvement of the quality of the software produced can also be a leverage argument

with their customers. Company heads are concerned with the logistical aspects of the system's implementation, particularly the technical constraints and the infrastructure to be made available.

**Toolsmiths.** The framework is mainly aimed at helping the creators of modeling assistant to define their system. To do so, they must respect the criteria of trust, creativity, the framework of multi-criteria recommender system. Moreover, in order to maximize the acceptability of their solution, they must respect the criteria of good assistant. They must set up the mechanisms to ensure the availability and updating of knowledge using the system, setting up the infrastructure.

| Concern | End-users | Client | Company head | Toolsmiths |
|---|---|---|---|---|
| **PURPOSES OF THE SYSTEM** | | | | |
| S.C. 1 | x | | | |
| **SUITABILITY OF THE ARCHITECTURE** | | | | |
| S.C. 2 | | | | x |
| S.C. 3 | x | | | x |
| S.C. 4 | | | | x |
| S.C. 5 | x | | | x |
| **FEASIBILITY OF CONSTRUCTING AND DEPLOYING** | | | | |
| S.C. 6 | | | | x |
| S.C. 7 | | | x | x |
| S.C. 8 | | | | x |
| **RISKS AND IMPACTS** | | | | |
| S.C. 9 | x | x | x | |
| S.C. 10 | x | x | x | |
| S.C. 11 | x | x | x | |
| S.C. 12 | x | x | x | |
| **MAINTAINABILITY AND EVOLVABILITY** | | | | |
| S.C. 13 | x | x | | x |
| S.C. 14 | x | x | | x |
| S.C. 15 | x | x | | x |

Table 7.2 – Association of stakeholders and concerns

## 7.6 Architecture viewpoints

The aim of architecture viewpoints is to answer the system concerns raised in Section 7.4. Each concern is addressed in one or more *archi-*

*tecture viewpoints*, as shown in Table 7.2, to ensure the suitability and robustness of the proposed architecture. In this section, we present and describe the three viewpoints of our framework as the *requirements viewpoint*, the *functional architecture viewpoint*, the *system requirements viewpoint*, and the *structural architecture viewpoint*.

### 7.6.1 Functional architecture viewpoint

In this section, we define the architecture from a functional viewpoint with the *architecture view* presented in Figure 7.2. This architecture view consists of a UML class diagram featuring the functional components of the system, whether they represent tasks, software, hardware, people, or concepts.

This view is centered on the *modeling assistant*, which is mainly composed of one or several *assistance features*. These features address one or several *modeling challenges* which emerge during one or several *modeling tasks*. The challenges may concern different aspects of modeling, which echoes S.C. 1. The modeling tasks are described by several *task characteristics* such as the time, the effort, the context understanding, the knowledge, or the new ideas that they require. The system helps the completion of these tasks with one or several assistance features, which then aim to reduce the modeling time (S.C. 9) or the modeling effort (S.C. 10), what constitutes its added value (S.C. 5.3).

From a functional point of view, the system supports creativity by embedding knowledge. Knowledge is made available (S.C. 3.1, S.C. 6) through the *knowledge base*, administered by one or several *stakeholders*. This is made possible by letting them providing *feedback* from the recommendations, or by importing external data or knowledge directly into the knowledge base. This mechanism ensures that new knowledge can explicitly integrated to the system or updated at any time (S.C. 13, S.C. 14).

Figure 7.2 introduces skills criteria that the system should own (S.C. 5.2). The assistant should be reputed for its *accuracy* and *performance* on performing one or several tasks. Its behaviour shall be predictable, so stakeholders know when to use it or not. The system adapts to the context of use (S.C. 5.5) by being integrated in the *modeling tool* and being able to analyze the *context*, composed of the *user intent*, the *task history*, and the *models* created by the stakeholders in the editor.

### 7.6.2 Structural architecture viewpoint

In this section, we define the architecture from a structural viewpoint, presented in Figure 7.3. This architecture view consists in a UML class diagram featuring the structure of a software assistant, based on three main parts, the *information repository*, the *knowledge back-end*, and the *user interface*.

The system assists modeling (S.C.1) by presenting *knowledge representations* to the user from a *knowledge display* that shall be a part of the *user interface*. These knowledge representations can be integrated into different *representation contexts* and fit different *representation forms* to support creativity (S.C. 3.2). This display adapts from the *context* (S.C. 5.5), according to the elements it gathers from the user interface through its *context collector*. It might also be tuned manually with *display filter controls*, to allow the users to customize their environment (S.C. 2.1, S.C. 3.3, S.C. 5.4), thus integrate better their modeling workflow (S.C. 11).

As identified in Chapter 4, the user interface shall own *usability characteristics* such as *action freedom* to let the user work without too much constraints, *workflow integration* to not disrupt the user while working, and shall ensure the *simplicity* of the interactions it features. This addresses usability concerns mentioned in S.C. 3.4 and S.C. 5.1. The user interface aims to be integrated in the modeling tool as a *modeling tool plugin* (S.C. 8), but may also be represented as an external interface, such as a standalone application. Then, the plugin must embed all components of the User Interface (UI) the *display filter controls* and the *trigger controls* views, the *knowledge display*, and the *context collector*.

The knowledge representations shall provide an *explanation* about the suggestions, which includes one or several *information trustworthiness characteristics* (S.C. 2.2). These characteristics enable the system to build a trust relationship with the users, as well as ensuring transparency (S.C. 3.4). Explaining why suggestions are proposed serves a mentoring purpose and might help users learn new insights about the domain or modeling good practices (S.C. 12).

The *information repository* is composed of a *knowledge base*, fed by a *data collector*, and managed through an *administration interface*. The data collector browses both *internal datasources* and *external datasources*, as configured by the system maintainers, to constitute the knowledge base (S.C. 3.1). Thus, at any time, knowledge can be integrated or updated to the system (S.C. 13, S.C. 14). This includes for instance the models edited by the employees of a company using the assistant and feeding a company-internal model repository.

The *knowledge back-end* is the component that exploits the knowledge

base, which provides input information for the production of knowledge useful to the user of the assistant. Note that the knowledge base contains *knowledge* when seen from their originating context. From the assistant point of view, it acts as *information* —which is not linked to a context —, which is transformed into knowledge by the *multi-criteria recommender system* (S.C. 3.1). The recommendations generated by the recommender algorithm are then exposed through an API, which guarantees knowledge accessibility (S.C. 6).

Several user control mechanisms are implemented throughout the components to ensure proper user adaptability (S.C. 2.1, S.C. 5.4). This enables a general tailorability of the system, which can be adapted to cover various needs and expectations (S.C. 3.3). The 3-tier architecture presented in this architecture view support the interoperability of the system (S.C. 15). Indeed, as multiple user interfaces can be plugged into the knowledge back-end, both back-end and information repository can be share among different assistants, to cover a broader number of assistance features.

### 7.6.3 Infrastructure viewpoint

In this section, we define the architecture from a infrastructure viewpoint, presented in Figure 7.4. This architecture view consists in a UML composite structure diagram exhibiting the infrastructure of a software modeling assistant.

The software assistant is based on a 3-tier architecture featuring an *information repository*, a *knowledge back-end*, and one or several *user interfaces*. In order to ensure interoperability, the information repository and the knowledge back-end should be deployed in two isolated servers (S.C. 3.4, S.C. 7). This enables the reusing of some components of the system to deploy other software assistants. It also augments the robustness and the reliability of the whole system. Software assistants can either share the same knowledge back-end, and then exploit the same recommendations and integrate them in their assistance features, or share the same knowledge. In the latter case, the two systems exploit different recommender systems to support different modeling assistance features.

### 7.6.4 System requirements viewpoint

This architecture view consists of a table, listing functional design requirements to build software assistants for software modeling. Based on the design constraints identified in Chapter 6, we provide a set of

requirements that either cover the whole system, or only apply to one specific component of our 3-tier structural architecture. These constraints are distributed as shown in Table 7.3, and are detailed in this section.

**S.R. 1. Information trustworthiness.** The modeling assistant shall be designed to express the *accuracy*, the *currency or novelty*, the *coverage or diversity*, the *attractiveness*, the *explicability*, the *believability*, and the *context compatibility* of information to perceive information trustworthiness.

**S.R. 2. System adaptability.** The modeling assistant shall enable *algorithm tunability*, *display tailorability*, and *data manageability* for the user to perceive system transparency and controlability.

**S.R. 3. Knowledge base.** The modeling assistants shall feature a knowledge base built around background knowledge, company-internal knowledge, and user-specific knowledge.

**S.R. 4. Knowledge availability.** The modeling assistant shall set up a system to guarantee the availability of knowledge. This may be achieved through the retrieval of edited models, but also from the analysis of work artifacts, and professional communications.

**S.R. 5. Knowledge representation choice.** The choice of the knowledge representation in the user interface of the modeling assistant shall be made according to the software engineer task that is aimed to be supported.

**S.R. 6. Knowledge representation switch.** The interface of the assistant shall allow the user to navigate between several representations of knowledge, in different forms.

**S.R. 7. Ease of knowledge access.** The modeling assistant shall allow a fast and easy access to the knowledge, and should allow a simultaneous display of several knowledge representations.

**S.R. 8. Tailorable environment.** The modeling assistant shall be adjustable according to the user's preferences and needs on the aspects of triggering, display, and internal behaviour.

**S.R. 9. User freedom.** The modeling assistant shall allow users to conduct their creative process as they like, and should not impose any constraints on the travels in and out the solution space.

**S.R. 10. Low threshold, high ceiling.** The modeling assistant shall be accessible to beginners, while allowing experts to perform their modeling tasks. The number of features of the assistant should be restricted to the very minimum, matching the requirements of one modeling task.

**S.R. 11. System interoperability.** The modeling assistant shall support interoperability, by being compatible with different modeling tools

| Scope | Requirements |
|---|---|
| Cross-component | • S.R. 1. Information trustworthiness (S.C. 2.2)<br>• S.R. 2. System adaptability (S.C. 2.1, S.C. 3.3)<br>• S.R. 11. System interoperability (S.C. 3.4)<br>• S.R. 18. Allocation of functions and tasks (S.C. 11) |
| Information repository | • S.R. 3. Knowledge base (S.C. 3.1)<br>• S.R. 4. Knowledge availability (S.C. 3.1) |
| Knowledge back-end | • S.R. 12. Object of decision (S.C. 4)<br>• S.R. 13. Criterion explainability (S.C. 4)<br>• S.R. 14. Criterion trustworthiness (S.C. 4)<br>• S.R. 15. Criterion context-accuracy (S.C. 4)<br>• S.R. 16. Aggregation function (S.C. 4)<br>• S.R. 17. Context adaptability for aggregation functions (S.C. 4) |
| User interface | • S.R. 5. Knowledge representation choice (S.C. 3.2)<br>• S.R. 6. Knowledge representation switch (S.C. 3.2)<br>• S.R. 7. Ease of knowledge access (S.C. 3.1)<br>• S.R. 8. Tailorable environment (S.C. 3.3)<br>• S.R. 9. User freedom (S.C. 3.3)<br>• S.R. 10. Low threshold, high ceiling (S.C. 3.4, S.C. 11)<br>• S.R. 19. Allocation of authority (S.C. 3.4, S.C. 11)<br>• S.R. 20. Allocation of responsability (S.C. 3.4, S.C. 11)<br>• S.R. 21. Allocation of resources (S.C. 3.4, S.C. 11)<br>• S.R. 22. Handover and takeover (S.C. 3.4, S.C. 11) |

Table 7.3 – Framework system requirements viewpoint

or different business domains.

**S.R. 12. Object of Decision.** The software assistant shall recommend elements (i) selected by their *Nature* or their presence in the hierarchy of elements from the *Model Scope*, represented in the *View Scope*, and tailored to the *Target* users.

**S.R. 13. Criterion explainability.** A recommendation criterion shall be described through a rationale that is easily understandable by the recommendation target.

**S.R. 14. Criterion trustworthiness.** A recommendation criterion shall reflect at least one of the following information trustworthiness characteristics: accuracy, currency/novelty, coverage/diversity, believability/explicability, context compatibility, and attractiveness.

**S.R. 15. Criterion context-accuracy.** A recommendation criterion shall exploit information from at least one of the four components of the object of decision.

**S.R. 16. Aggregation function.** The aggregation function of the multi-criteria recommender system shall be defined as follows

$$\text{Let } (w_1, w_2, ..., w_n) \in [0;1]^n \text{ where } \sum_n w_n = 1,$$

$$\begin{array}{rccc} overall & : & \mathcal{A} & \rightarrow & [0;1] \\ & & a & \mapsto & \sum_n w_n \times s_n \end{array} \tag{7.1}$$

with $s_1, s_2, ..., s_n$ the individual scores for the $n$ criteria
$w_1, w_2, ..., w_n$ the weights for the $n$ individual scores
$\mathcal{A}$ the set of recommender alternatives

**S.R. 17. Context adaptability for aggregation functions.** The aggregation function of the recommender system shall be defined for each context $k$ as follows

$$overall_k(a) = \sum_n w_{n,k} \times s_n \tag{7.2}$$

**S.R. 18. Allocation of functions and tasks.** When conceiving the software system, the designer shall clearly define which functions will be allocated to the system and which tasks will be allocated to the user.

**S.R. 19. Allocation of authority.** When conceiving the software system, the designer shall clearly identify which entity is allowed to trigger or prevent functions/tasks execution.

**S.R. 20. Allocation of responsibility.** When conceiving the software system, the designer shall clearly identify which entity is responsible for

the outcome of the execution of the functions/tasks.

**S.R. 21. Allocation of resources.** When conceiving the software system, the designer shall clearly identify which resource (e.g., diagrams, preferences, documents) is allocated to which entity in terms of production, modification or sharing with the other entity.

**S.R. 22. Handover and takeover.** When designing the system, control transitions sequences shall be defined clearly, as well as which entity can trigger them.

## 7.7 Correspondence rules

This section introduces the correspondence rules between system concerns and architecture views, as presented in Table 7.4. Anwsers to each system concerns are provided in the identified architecture views.

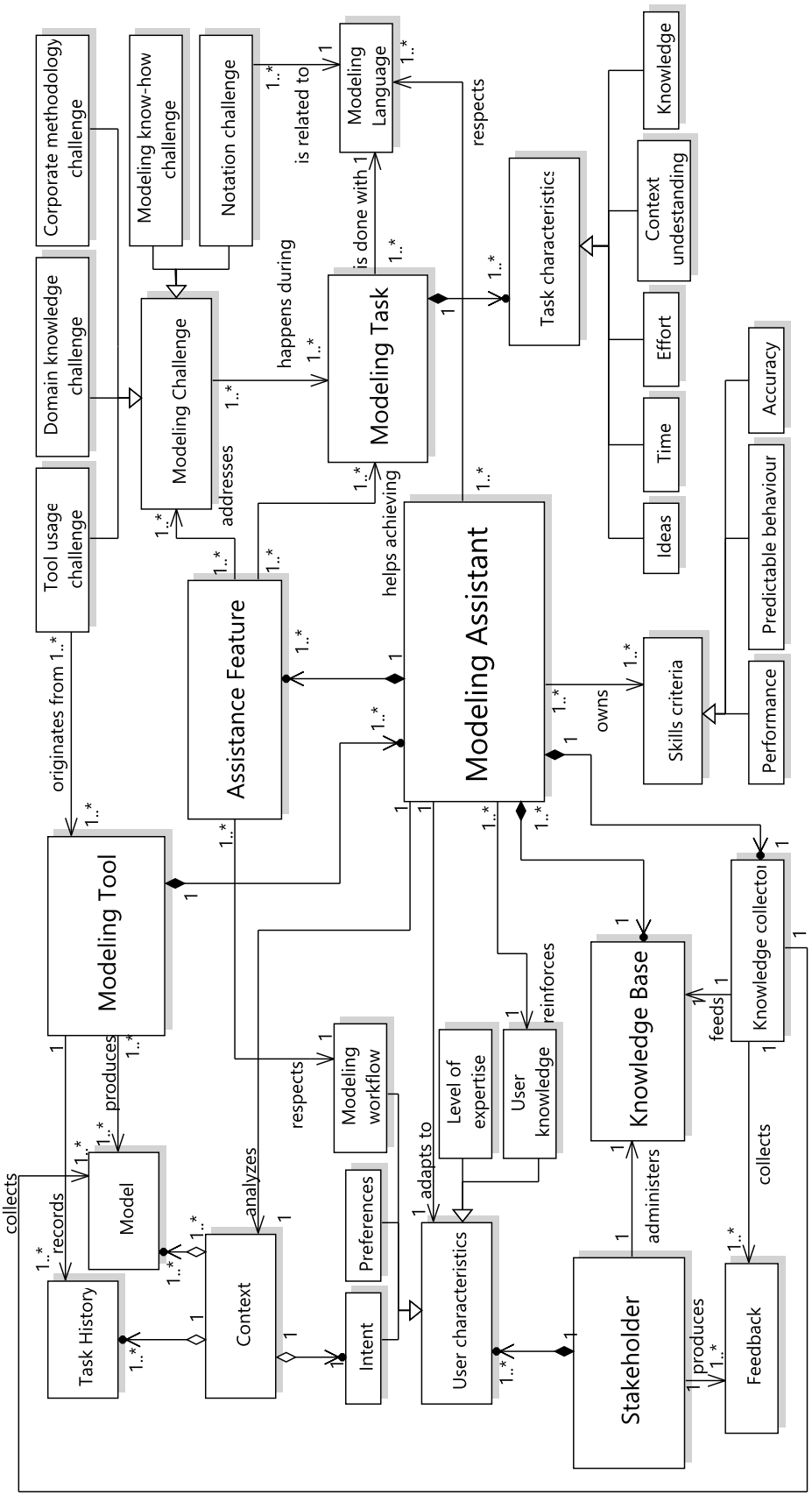



Figure 7.2 – Framework functional viewpoint

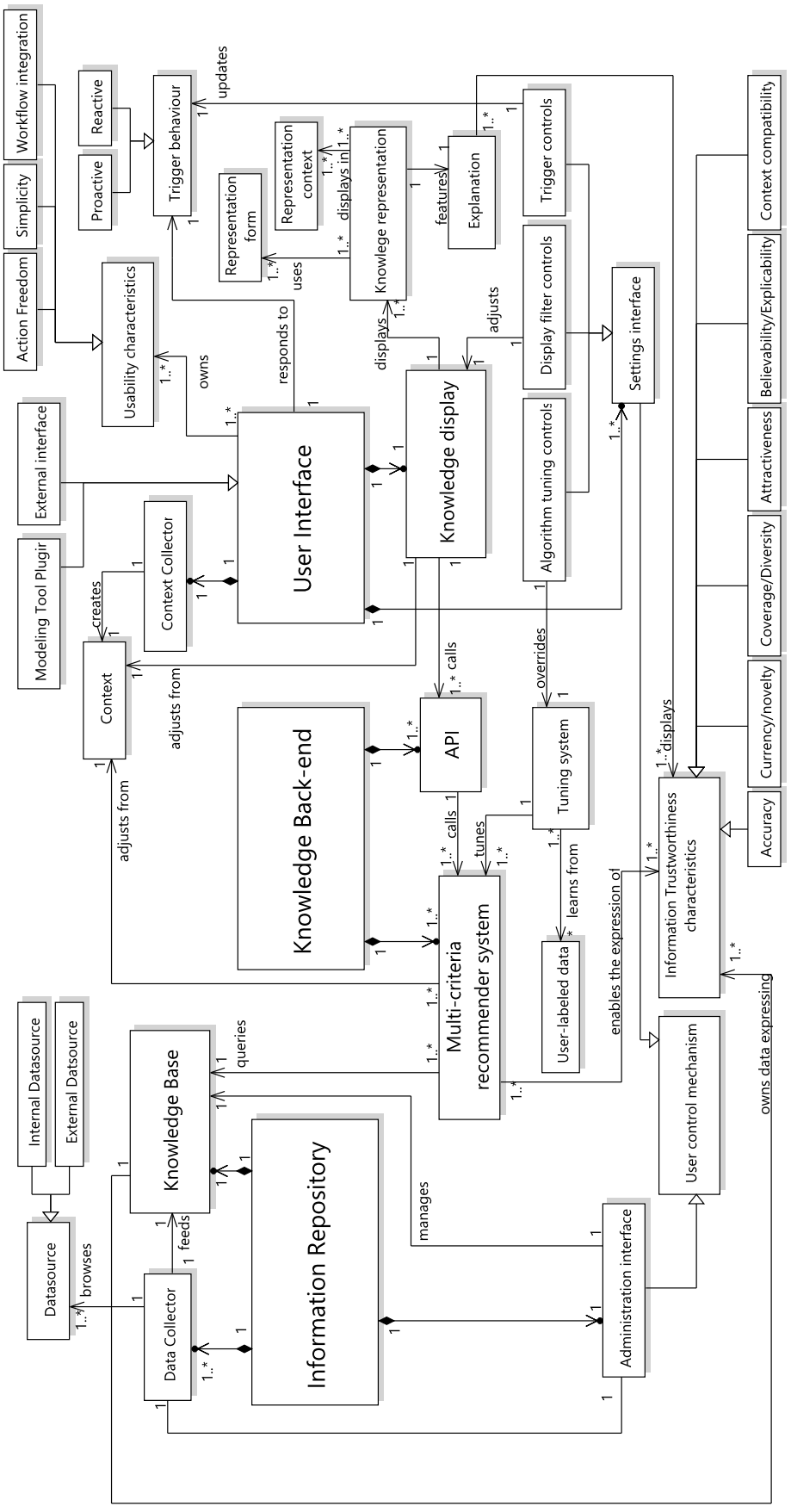



Figure 7.3 – Framework structural viewpoint

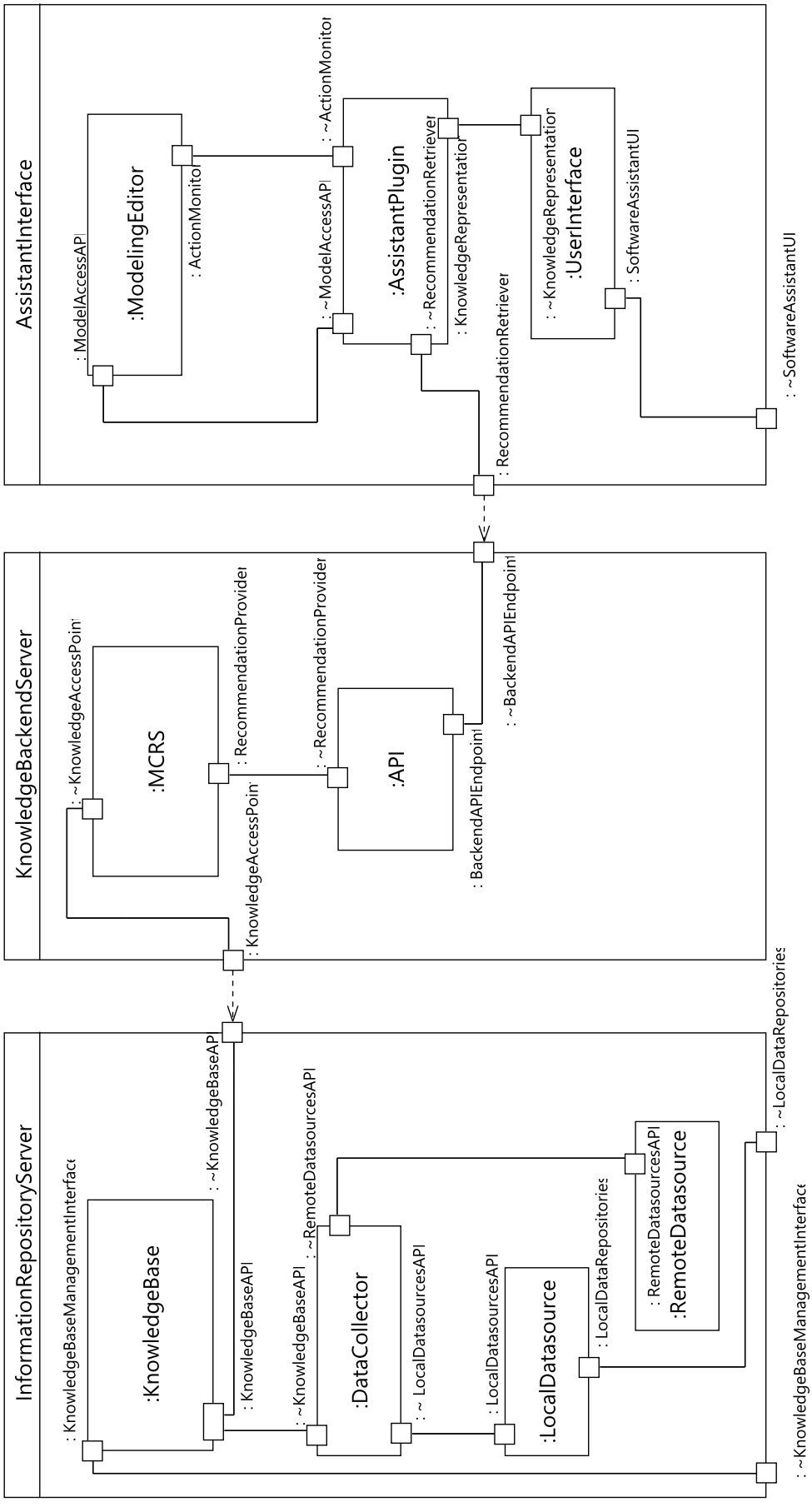



Figure 7.4 – Framework infrastructure viewpoint

| Concern | Functional | Structural | Infrastructure | Requirements |
|---|---|---|---|---|
| **Purposes of the system** | | | | |
| S.C. 1 | x | x | | |
| **Suitability of the architecture** | | | | |
| S.C. 2.1 | | x | | x |
| S.C. 2.2 | | x | | x |
| S.C. 3.1 | x | x | | x |
| S.C. 3.2 | | x | | x |
| S.C. 3.3 | | x | | x |
| S.C. 3.4 | | x | x | x |
| S.C. 4 | | | | x |
| S.C. 5.1 | | x | | |
| S.C. 5.2 | x | | | |
| S.C. 5.3 | x | | | |
| S.C. 5.4 | | x | | |
| S.C. 5.5 | x | x | | |
| **Feasibility of constructing and deploying** | | | | |
| S.C. 6 | x | x | | |
| S.C. 7 | | | x | |
| S.C. 8 | | x | | |
| **Risks and impacts** | | | | |
| S.C. 9 | x | | | |
| S.C. 10 | x | | | |
| S.C. 11 | | x | | x |
| S.C. 12 | | x | | |
| **Maintainability and evolvability** | | | | |
| S.C. 13 | x | x | | |
| S.C. 14 | x | x | | |
| S.C. 15 | | x | | |

Table 7.4 – Correspondence between system concerns and architecture viewpoints

# Part III

# Validating our approach: preliminary work and discussion

Chapter 8

# Designing a software modeling assistant

*Chapter 7 defines a framework to develop software assistants for software modeling. As a first effort in validating this framework, we follow its approach for designing one modeling assistant. More specifically, we follow the guidelines to build a system that is able to suggest class attributes and relationships for UML class diagrams within the Papyrus modeling tool*[1]*. In this chapter, we describe the system according to the four architecture views, we provide an overview of how the modeling assistant addresses system design concerns, and we detail the design of the multi-criteria recommender system.*

## 8.1 Modeling assistant design

As a first effort in validating the framework, we define one specific modeling assistant. We decided to address UML modeling as it is a topic we investigated in our interviews from Chapter 4. More particularly, the assistant focuses on the creation on UML class diagrams, as it is the most used representation in the UML ecosystem [136].

From all the language elements belonging to class diagrams, we decided to address the creation of classes and class attributes. The assistants consists in a recommender for UML class diagrams, which

[1] https://www.eclipse.org/papyrus/

suggests new ideas of classes and attributes to add in the diagram, and eventually adds them if requested. Therefore, the system mainly provides *conceptual assistance*, focusing more on the mastery of the business domain than on the mastery of the modeling language or tool. As a reminder, this assistance feature was *the most* requested by the modeling experts in the interviews conducted in Chapter 4.

In this section, we describe the modeling assistant from the four architecture *functional*, *structural*, *infrastructure*, and *requirements* views defined in the formal framework. We also provide an overview of how the assistant addresses the system concerns identified in the framework.

### 8.1.1 Functional architecture description

In this section, we exploit the *functional architecture viewpoint* from the formal framework. We describe how each concept presented in this artifact is reflected in the design of our instance of modeling assistant.

#### Assistance feature

The overall task of creating UML class diagrams embeds a large number of sub-tasks, such as the following:

1. Perform ideation to identify suitable classes to add to the diagram
2. Perform ideation to identify suitable attributes to add to a specific class
3. Perform ideation to identify suitable relationships between classes
4. Manually add a class to the diagram, by selecting the class creation tool, drawing it, and typing its name.
5. Manually add an attribute to a specific class, by selecting the attribute tool, selecting the class, and typing all required attribute information
6. Manually add a relationship between two existing classes, by selecting the proper tool for the relation type, and selecting the source and target classes
7. Layout the classes in the diagram
8. Layout the relationships connectors, with their names and multiplicities

Our system aims at providing two main assistance features which consist in recommending (i) class attributes and (ii) class relationships for a UML class diagram. These two features address the sub-tasks 1 to 6 listed previously, which require cognitive effort related to ideation, and time to create elements and perform ideation.

#### Knowledge base

The knowledge base is created from the models extracted from the GenMyModel public repositories[2].

#### Behaviour

The assistant is not able to delete elements from the model other than ones it created to ensure that it does not interfere with user intents. It has write permission on the diagram but only adds elements on the explicit request from the user to do so.

#### Context

The system collects information about the currently edited model, and monitors the classes selected in the editor to provide dedicated recommendations. The assistant retrieves the whole model from the editor and provides it to the knowledge back-end when requesting recommendations.

### 8.1.2 Structural architecture description

In this section, we describe how each of the three components presented in the *structural architecture viewpoint* of the framework is reflected in the design of our instance of modeling assistant.

#### User interface

The user interface will be integrated within the Papyrus modeling tool, and is accessible through a key binding ⇧+←. It features a knowledge display representing the currently edited class, with a list of recommendations for class attributes and class relationships, and an explanation interface. The displayed knowledge is then filtered according to the edited class. Users might adjust the detail of information that is displayed, to get more or less details about the suggestions.

[2] https://www.genmymodel.com

The assistant does not constrain the users in their actions, as it starts only when explicitly requested. Moreover, users can adjust the recommendations before adding them, by changing the type or the multiplicity of the class attributes, or by changing the type of the relationship. The assistant mainly relies on three basic interaction inputs with (i) a key binding to start the system, (ii) a mouse scroll to browse the recommendation lists, (iii) clicks to switch between recommendation lists, and to add a recommended element to the model.

The assistant features an explanation interface, describing the rational behind the individual recommendation scores from the recommendation algorithm. It displays contextualized information about statistics that were computed to perform ranking and sorting. This explanation addresses aspects of accuracy, transparency, believability, and explainability. The nature of the individual scores expresses currency/novelty, and context compatibility.

#### Information repository

The data collector of our assistant gathers models from the GenMyModel repositories, as well as local models stored in a dedicated folder. These models are mapped to a graph and stored in a graph database, that provides API access, and a user interface for management.

#### Knowledge back-end

The knowledge back-end consists of two four-criteria recommender systems that respectively act for attributes and relationships recommendations. They are both exposed on a same API endpoint that enables the retrieval of modeling recommendations. The weights of the aggregation functions can be set manually, or learned with machine learning from user-labelled data.

The tailorability of the system is ensured by (i) the data management interface provided by the information repository, (ii) the weight setup mechanism in the recommender system, and (iii) the display filter in the user interface.

### 8.1.3 Infrastructure description

The system will be deployed on one server, hosting the information repository and the knowledge back-end. This choice is guided by cost constraints, and hence limits the robustness of the system. Consequently,

the knowledge base will be unavailable to knowledge back-ends other than the one designed for the prototype.

### 8.1.4 System requirements description

In this section, we detail the characteristics of the system over each of the 22 system requirements from the framework defined in Chapter 7.

**S.R. 1. Information trustworthiness.** Information trustworthiness characteristics are expressed through the user interface in the Papyrus plugin. This explanation provided in the knowledge display addresses aspects of accuracy, transparency, believability, and explainability. The nature of the individual scores expresses currency/novelty, and context compatibility, as described in Section 8.2.

**S.R. 2. System adaptability.** Users of the modeling assistant can adapt the system to their needs by integrating their own models into the knowledge base, by influencing the aggregation functions of the recommender based on data that they can label, and by choosing the information they want to display in the Papyrus plugin.

**S.R. 3. Knowledge base.** The assistant exposes a knowledge base embedding models from several anonymous users.

**S.R. 4. Knowledge availability.** The knowledge base contains more than 100,000 models, retrieved from the GenMyModel public repositories.

**S.R. 5. Knowledge representation choice.** The recommendations are provided through the knowledge display under different representations. They are first grouped in two lists for attributes and relationships. Each representation is also detailed in the explanation display with rationales about the reason why it is being suggested.

**S.R. 6. Knowledge representation switch.** The user can easily switch between the list representation of suggestions and the detail view of each recommendation. The explanation is accessible in the same view, by hovering over the elements in the list.

**S.R. 7. Ease of knowledge access.** Accessing the recommendations requires 2 actions from the users. They must first select the class that they want recommendations for, and start the assistant by triggering the keyboard shortcut.

**S.R. 8. Tailorable environment.** The modeling assistant features filters to show or hide details about each recommendation. This mechanism allows users to make the display lighter or more complex as they wish.

**S.R. 9. User freedom.** As it is triggered manually, the assistant does not constrain users to a specific workflow. Thus, users can work the way

they are used to, and benefit from the assistance feature whenever they want, without any previous disruption.

**S.R. 10. Low threshold, high ceiling.** The modeling assistant provides lists of recommendations for class attributes and relationships, and integrates the UML concepts of multiplicity, types, and relation types. Thus, it is accessible to users with a basic understanding of the UML syntax. It proposes two main assistance features which remain only focused on the edition of a UML class in the modeling tool. In the meantime, modeling experts might use these features in their tasks, as clearly indicated in Chapter 4.

**S.R. 11. System interoperability.** The modeling assistant owns knowledge from thousands of models that cover a broad variety of domains and concepts. Multiple user interfaces can connect to the exposed knowledge back-end API. This allows for the creation of different assistance features exploiting the same recommendations, or the integration of the same assistance features in other modeling tools.

**S.R. 12, 13, 14, 15, 16, 17. Recommender system** These requirements are addressed in Section 8.2.

**S.R. 18. Allocation of functions and tasks.** The user is responsible for opening the user interface of the system, selecting accurate recommendations to add to the model, triggering the add action, and closing the user interface. The system is responsible for starting when requested, retrieving context elements to compute recommendations, computing recommendations, displaying it to the user, and adding elements to the model when requested.

**S.R. 19. Allocation of authority.** The user has full authority to turn the system on and off, and has the choice to continue or stop to use the system at any time.

**S.R. 20. Allocation of responsibility.** The user is fully responsible for the outcomes of the use of the software assistant to edit models. The recommendations are provided for informational purposes, and do not necessarily reflect best practices or the design direction to be explored.

**S.R. 21. Allocation of resources.** The software assistant has full write permissions on the edited model, in order to perform the requested edition operations. It monitors the elements selected in the active modeling editor of the modeling tool, to contextualize its recommendations.

**S.R. 22. Handover and takeover.** When requesting to add or remove an element from the model, the modeling assistant takes over the control of the modeling tool. It can not be stopped during this edition operation. When completed, it hands over the control to the user, who can continue to use the system or stop it.

### 8.1.5 System concerns overview

Table 8.1 summarizes the design solutions provided in the four architecture descriptions, to build an overview of our modeling assistant.

| System concern | Design solution |
|---|---|
| S.C. 1. How does the system assist modeling? | The system provides two main assistance features by suggesting class attributes to add for a specific class in a UML class diagram, and suggest class relationships to add in a UML class diagram. It offers the possibility to add suggested elements automatically to the model if requested. |
| S.C. 2.1. How does the system enable user control? | The system allows user to integrate their own models into the knowledge base. Its behaviour can be adjusted by modifying the weights of the recommendation aggregation functions manually, or through machine learning based on user-custom labelled data. The user interface of the assistant can be arranged to show or hide some details about the recommendations. |
| S.C. 2.2. How does the system handle information-trustworthiness characteristics? | The system features an explanation interface which is mainly responsible for expression of the accuracy, the transparency, the believability, and the explainability of each suggestion. The nature of the individual recommendation criteria expresses currency/novelty. Some of these characteristics are also expressed through the use of visual confidence score indicators displayed for each item in the recommendation lists. |
| S.C. 3.1. How does the system ensure knowledge availability? | The system collects the models from the public repositories of GenMyModel, an online modeling tool, which contain more than 100,000 models. These models are integrated to the knowledge with those specifically defined by the user, to ensure data manageability. |
| S.C. 3.2. How does the system generate knowledge representations? | The system monitors the actions of the user to determine the selected elements in the editor. When a class is selected, the assistant plugin requests recommendation from the knowledge back-end. These recommendations are displayed upon user request and presented in the knowledge display. Knowledge is represented as list of recommendations, and as explanations for each element of the list. Visual indicators are also used to provide another knowledge representation to contextualize each considered recommendation. |
| S.C. 3.3. How does the system adapt to users' paths and styles? | The system does not impose any change on the workflow to be operated. It starts upon user request and provides recommendation in a asynchronous manner. When installed, the assistant may not be used, and is transparent to the user in that case. Its behaviour can be tailored according to the description of S.C. 2.1. It proposes different knowledge representations as detailed in S.C. 3.2. |

| | |
|---|---|
| S.C. 3.4. How does the system achieve usability, transparency, and interoperability? | The system relies on a keyboard shortcut to be triggered, and relies on 2 major mouse interactions *click* and *scroll*. The shortcut should be clearly reminded to the user to ensure usability. Being manually triggered, the system does not disrupt the user with unwanted solicitations. The transparency is achieved through the explanation interface which details the reason why each suggestion is included in the recommendation list. Our assistant can operate over different business domains, and can be extended with different assistant plugins from different modeling tools, which supports interoperability. |
| S.C. 4. How does the system implement a multi-criteria recommender system? | Detailed in Section 8.2. |
| S.C. 5.1. How does the system demonstrate usability? | See S.C. 3.4. |
| S.C. 5.2. What skills does the system have? | The system implements a multi-criteria recommender system to produce recommendation that fit best the modeling context and users' needs. It has been tested so it does not unexpectedly edit the diagrams. Results of the recommender algorithm evaluation are provided in Chapter 10. |
| S.C. 5.3. What is the added value of the system? | The modeling assistant aims at identifying potentially missing model elements for both modeling beginners and experts. It provides new design ideas by suggesting relationships and attributes, and thus supports creativity through the evocation process. As the system is able to add element to the model, it might help save the required time and effort to produce these elements manually. |
| S.C. 5.4. How does the system adapt to the user? | See S.C. 3.3. |
| S.C. 5.5. How does the system adapt to the context? | The system monitors the actions of the user to determine the selected elements in the editor. It exploits the content of the currently edited model, which is transmitted to the knowledge back-end to produce recommendations. |
| S.C. 6. How does the system ensure data accessibility? | The modeling assistant exposes the knowledge back-end API, which enables any client to retrieve modeling recommendations about UML class attributes and UML class relationships. |
| S.C. 7. What is the required infrastructure to operate the system? | The modeling assistant information repository and knowledge back-end are hosted on a remote server, which is requested by the assistant clients. |
| S.C. 8. How can the system be integrated in a modeling tool? | The assistant clients are integrated into the Papyrus modeling tools as Eclipse plugins, reacting to actions performed in the active Papyrus editor. |

| | |
|---|---|
| S.C. 9. How does the system aim to reduce the modeling time? | The modeling assistant provides new ideas for the design of software solutions as modeling recommendations. This might help reduce the time spent on the ideation process of the modeling task. In the meantime, the assistant is able to add elements to the model, that might take time to create. Hence, it may reduce the time to create model elements. These enhancements shall however be evaluated through empirical evaluations. |
| S.C. 10. How does the system aim to reduce the modeling effort? | See S.C. 9. |
| S.C. 11. How does the system integrates to the user's modeling workflow? | See S.C. 3.3. |
| S.C. 12. How does the system can help users learn? | The modeling assistant provides explanations about the reason why each alternative is presented in the recommendation list. Such explanations might help the user learn about domain rules, modeling good practices, or other design approaches. |
| S.C. 13. Can new knowledge be integrated to the system at any time? | The data collector of the modeling assistant retrieves models from the GenMyModel public repositories and from a specific local folder defined by the user. Hence, at any time, users can add new models in the knowledge base, by starting the data collector process. |
| S.C. 14. Can knowledge be updated in the system at any time? | By triggering the data collector application, the knowledge base is rebuilt from scratch. Thus, its content can be updated easily by updating the models and starting the data collection process. |
| S.C. 15. Can system knowledge be retrieved from new/multiple software clients? | The architecture of the modeling assistant is based on client-server communications. Thus, as stated in S.C. 3.4, the knowledge back-end can be requested from different clients regardless of their origin. |

Table 8.1 – Modeling assistant design overview from the concerns perspective

## 8.2 Designing the multi-criteria recommender system

Our modeling assistant relies on a recommendation engine that suggests UML class relationships and class attributes. To define this recommender system, we exploit the guidelines provided in the formal framework. It introduces several requirements about the design of the assistant's recommender, that guides its definition. Thus, we define the object of decision, the family of criteria, and the utility function following the framework guidelines.

In this section, we first introduce a formal background for criteria description. Then, we describe the object of decision, and we provide formal descriptions of the four criteria that compose our multi-criteria recommender system for recommending UML class attributes for UML classes. In this manuscript, we do not describe the recommender for UML class relationships which mimics the behaviour of this recommender system.

### 8.2.1 Formal background

To emphasize information trustworthiness and transparency using the Multi-Criteria Rating Recommender Systems (MCRS) construction methodology [4], we define a criterion as a combination of the following three elements: (i) a rationale, (ii) a selection filter, and (iii) a ranking function. Note that, since our system is rating-based, the ranking function is a rating function.

#### Preliminary concepts

The Recommendation Class ($RC$) is the class for which attribute recommendations are to be provided. The following sets are defined:

- $\mathcal{M}$ the set of all models in the database,
- $\mathcal{M}(c)$ the singleton model that owns class $c$. Note that $\mathcal{M}(c) = \{M(c)\}$, where $M(c)$ is the only model that owns class $c$.
- $|M(c)|$ the set of all class names of classes in $M(c)$.
- $\mathcal{C}$ the set of classes from all models in $\mathcal{M}$.
- $\mathcal{C}_{|c|}$ the set of classes from all models in $\mathcal{M}$ that have the same name as $c$.
- $\mathcal{C}(a), a \in \mathcal{A}$ the singleton containing the only class that owns attribute $a$. Note that $\mathcal{C}(a) = \{C(a)\}$, where $C(a)$ is the only class that owns attribute $a$.
- $\mathcal{C}(|a|)$ the set of classes that own an attribute that has the same name as $a$. By extension, we denote the set of classes that own attributes with the same name by $a_1, a_2, ..., a_n$. $\mathcal{C}(|a_1|, |a_2|, ..., |a_n|)$ .
- Let $X \subset \mathcal{C}$. Then $|X|$ is the set of the names of classes in $X$.
- $\mathcal{A}$ the set of attributes of all classes in $\mathcal{C}$.

- $\mathcal{A}_c$ the set of attributes owned by class $c$.
- Let $X \subset \mathcal{A}$. Then $|A|$ is the set of the names of attributes in $X$.

### Rating functions

According to the constraints on a family of criteria, we define the rating function $r$. Let $a$ be a candidate attribute for $RC$, $crit$ the criterion specified by $r$. We define $r_{crit,RC}$ as follows:

$$\begin{array}{rcccc} r_{crit,RC} & : & |\mathcal{A}| & \rightarrow & [0;1] \\ & & |a| & \mapsto & r(|a|) \end{array} \tag{8.1}$$

We also define the following constraint on the nature of $r_{RC}$. Let $a_1$, $a_2$ be two attribute candidates for $RC$. Then

$$\begin{array}{c} a_1 \text{ performs better than} \\ a_2 \text{ on } crit \text{ for } RC \end{array} \quad \Leftrightarrow \quad r_{crit,RC}(|a_1|) > r_{crit,RC}(|a_2|) \tag{8.2}$$

## 8.2.2 Object of decision and criteria identification

### Define the object of decision

The System Requirement 12 from the *system requirement viewpoint* of the framework recommends the description of the *object of decision* of the recommender system as follows.

**S.R. 12. Object of Decision.** The software assistant shall recommend elements (i) selected by their *Nature* or their presence in the hierarchy of elements from the *Model Scope*, represented in the *View Scope*, and tailored to the *Target* users.

Our modeling assistant aims to recommend attributes for classes in UML class diagrams. Thus, the nature of the alternatives to be suggested is *attributes*. The model scope refers to the direct or indirect owners of attributes, and then map to *classes or packages*. The view scope is the UML views in which the alternatives are represented, which corresponds to *UML class diagrams*. Finally, the target users are *users of the modeling tool, who may have previous experience, personal preferences, and various levels of propensity to trust*. Consequently, we define the object of decision of our recommender system as follows

**Modeling assistant's object of decision.** The software assistant recommends attributes (i) selected by their nature or their presence in the hierarchy of classes or packages, (ii) represented in class diagrams, and (iii) tailored to the user of the modeling tool, who may have previous

experience, personal preferences, and various levels of propensity to trust.

### Identifying candidate criteria

The System Requirement 13 from the *system requirement viewpoint* of the framework requires each criterion to be translatable to a text rationale. Then, the requirement 14 states that each criterion shall reflect at least one of the listed information trustworthiness characteristics. Finally, the requirement 15 imposes each criterion to exploit information from at least one of the four component of the object of decision.

To identify criteria that might fit the purpose of our recommender system to suggest UML class attributes, we combine these three requirements as one table. Table 8.2 presents the rationales for the nine candidate criteria that were identified by applying this methodology. In this table, lines are defined by each of the characteristic of information trustworthiness, columns are determined by each of the four components of the object of decision, and cells are filled with potential recommendation criteria.

Ideally, all of the criteria should be selected for determining the final recommendations set. However, practice has demonstrated that collecting all required data in a systematic, reliable and automated manner is impractically complex. Multiple prior works [41] [182] outline these challenges and validate the common practice of selecting a coherent subset from all possible criteria according to the goals of the system. Based on the data we have at hand and for the first prototype, we decide to implement criteria based on rationales CR2, CR3, CR4 and CR5 respectively as criteria C1, C2, C3, and C4. In the next sections, we detail the rationale and the score formula for each of these four criteria.

## 8.2.3 In-class recurrence criterion (C1)

We define criterion C1, which refers to in-class recurrence, based on criterion rationale CR2. In the following, the term 'owner class' of an attribute refers to the class that directly owns that attribute.

**Rationale.** The attribute is often present in classes with the same name as the owner class.

**Selection filter.** Attributes owned by classes with the same name as *RC*.

**Rating approach.** The most frequently occurring candidate gets the highest score and the least frequently occurring candidate gets the

Table 8.2 – Criteria identification grid

| | Nature | Model Scope | View Scope | Target |
|---|---|---|---|---|
| **Accuracy** | Ø | **CR1**: The attribute is semantically close to the concept represented by the owner class (i.e., the class that owns the attribute). | Ø | Ø |
| **Currency**, **Novelty** | Ø | **CR2**: The attribute is often present in classes named as the owner class.**CR3**: The attribute is only present in classes named as the owner class. | Ø | Ø |
| **Coverage**, **Diversity** | Ø | Coverage results from the definition of a low enough acceptability threshold on measurable scores based on each defined criterion so that a reasonably broad range of quality-varying recommendation are provided. | | |
| **Believability**, **Explicability** | Ø | Ø | Ø | **CR7**: The attribute is often used by trusted persons or in trusted repositories to describe the owner class. |
| **Context Compatibility** | Ø | **CR4**: The attribute is often present with the other attributes of the owner class. **CR5**: The attribute often describes a class named as the owner class in similar models. | **CR6**: The attribute often describes a class named as the owner class in similar class diagrams. | **CR8**: The attribute fits an end-user's experience, desires and moods (be surprised, be efficient, ...). |
| **Attractiveness** | Ø | Ø | Ø | **CR9**: The attribute meets an end-user's formatting preferences. |

lowest non-null score. Candidates that do not appear in a class with the same name as the owner class get a null score.

**Rating function.** Let $occ$ be a function that, for attribute $a$ in class $c$, returns $occ_{|c|}(|a|)$ the occurrence number of attributes named $|a|$ in classes named $|c|$. Let $card(X)$ denote the cardinality of set $X$, then

$$occ_{|c|}(|a|) = card(\{\ a \mid |a| \in |\mathcal{C}_{|c|}|\ \}) \tag{8.3}$$

Let $occ(|a|)$ be the number of occurrences of attributes named $|a|$ in all classes of $\mathcal{C}$:

$$occ(|a|) = card(\{\ a \mid |a| \in |\mathcal{C}|\ \}) \tag{8.4}$$

Finally, we define $r_{C1,RC}$ as follows:

$$r_{C1,RC}(|a|) = \begin{cases} \dfrac{occ_{|RC|}(|a|)}{\max\limits_{b \in \mathcal{C}(|a|)} \left( occ_{|RC|}(|b|) \right)} & \text{if } \mathcal{C}(|a|) \neq \emptyset \\ 0 \text{ else} \end{cases} \tag{8.5}$$

### 8.2.4 In-class exclusivity criterion (C2)

We define criterion C2, which refers to in-class exclusivity, based on criterion rationale CR3.

**Rationale.** The attribute is only present in classes with the same name as the owner class.

**Selection filter.** Attributes owned by classes with the same name as *RC*.

**Rating approach.** Candidates which only appear in classes with the same name as the owner class get the highest score. Those that appear equally in all classes of the data set get the lowest score. Candidates that never appear in a class with the same name as the owner class get a null score.

**Rating function.** We then define the in-class exclusivity rating function as follows:

$$r_{C2,RC}(|a|) = \begin{cases} \dfrac{occ_{|RC|}(|a|)}{occ(|a|)} & \text{if } \mathcal{A} \neq \emptyset \\ 0 \text{ else} \end{cases} \tag{8.6}$$

### 8.2.5 Attribute synergy criterion (C3)

We define criterion C3, which refers to attribute synergy, based on criterion rationale CR4.

**Rationale.** The attribute is often present along with other attributes of the owner class.

**Selection filter.** Attributes connected to attributes owned by *RC* through their presence in a common class. Common classes are those classes that share the same name.

**Rating approach.** The more often a candidate and an attribute of *RC* appear together in a class, the higher the score. The more that a candidate appears together with different attributes of *RC*, the higher the score. Candidates which never appear together in a class get a null score.

**Rating function.** We define the attribute synergy rating function as follows:

$$r_{C3,RC}(|a|) = \begin{cases} \dfrac{\sum\limits_{b \in \mathcal{A}_{RC}} \Big( card(\mathcal{C}(|a|,|b|)) \Big)}{card\Big(\mathcal{A}_{RC}\Big) \cdot card\Big( \bigcup\limits_{b \in \mathcal{A}_{RC}} \mathcal{C}(|a|,|b|) \Big)} \\ \text{if } \bigcup\limits_{b \in \mathcal{A}_{RC}} \mathcal{C}(|a|,|b|) \neq \emptyset \text{ and } \mathcal{A}_{RC} \neq \emptyset \\ \\ 0 \text{ else} \end{cases} \tag{8.7}$$

### 8.2.6 Context similarity criterion (C4)

We define criterion C4, which refers to context similarity, based on criterion rationale CR5.

**Rationale.** The attribute often describes a class named the same as the owner class in similar models.

**Selection filter.** Attributes owned by classes named $|RC|$ in models which share at least two common classes with $M(RC)$.

**Rating approach.** Candidates from models that share the highest number of classes with $M(RC)$ get the highest score. Candidates from models which have no class in common get a null score.

**Rating function.** We define $com_{RC}(a)$ as the number of classes other than $RC$ that are common to the model owning candidate $a$ and $M(RC)$.

$$com_{RC}(a) = card\Big( \, |M(C(a))| \cap |M(RC)| \, \Big) - 1 \tag{8.8}$$

The candidate set can now be defined as follows:

$$cand_{C4}(RC) = \Big\{ \, a \in \mathcal{A} \, \Big| \, |C(a)| = |RC|, com_{RC}(a) > 0 \, \Big\} \tag{8.9}$$

Next, we define $context_{RC}(|a|)$ as the total number of classes in common between $M(RC)$ and all models containing a class named $|RC|$ that owns an attribute named $|a|$.

$$context_{RC}(a) = \sum_{\substack{|p|=|a| \\ p \in cand_{C4}(RC)}} com_{RC}(p) \tag{8.10}$$

Finally, for normalisation purposes (as required by rating function constraints) we define

$$r_{C4,RC}(|a|) = \begin{cases} \dfrac{context_{RC}(a)}{\max\limits_{b \in cand_{C4}(RC)} (context_{RC}(b))} & \text{if } cand_{C4}(RC) \neq \emptyset \\ \\ 0 \text{ else} \end{cases} \tag{8.11}$$

### 8.2.7 Utility Function

The framework that we exploit formally defines the aggregation function of the recommender system through System Requirement 16. It consists in the weighted sum of all considered criteria. Thus, we define the utility as follows:

Let $(a, b, c, d) \in [0;1]^4$ where $a + b + c + d = 1$,

$$\begin{array}{rcccl} overall_{RC} & : & |\mathcal{A}| & \rightarrow & [0;1] \\ & & |p| & \mapsto & a \times s_1 + b \times s_2 + c \times s_3 + d \times s_4 \end{array} \tag{8.12}$$

$$\text{with} \quad \begin{array}{l} s_1 = r_{C1,RC}(|p|), s_2 = r_{C2,RC}(|p|), \\ s_3 = r_{C3,RC}(|p|), s_4 = r_{C4,RC}(|p|) \end{array}$$

The machine-learning process is used to determine the values of the weights $a$, $b$, $c$, and $d$.

#### Context adaptability

In the context of recommending UML attributes for classes, we identify four different possible *Contexts*. The system will provide recommendations for the following:

- **Context 1:** A class owning no attributes and no other classes in the model.
- **Context 2:** A class owning one or more attributes and no other classes in the model.
- **Context 3:** A class owning no attributes in a model but containing one or more other classes.
- **Context 4:** A class owning one or more attributes in a model and also containing one or more other classes.

Each of the above contexts has access to different information so that not all of the criteria can be applied equally to all of them. For instance, the context similarity criterion C4 relies on the presence of other classes in the diagram and is, therefore, not applicable to contexts 1 and 2. Consequently, we define the overall utility function in context k $overall_{k,RC}$ as:

$$overall_{k,RC}(|p|) = a_k \times s_1 + b_k \times s_2 + c_k \times s_3 + d_k \times s_4 \tag{8.13}$$

This results in four different utility functions corresponding to the four different contexts. They are determined individually in the course of the machine-learning process.

### Utility function determination

The aggregation functions require their weights to be defined to compute overall recommendation scores. However, these scores can hardly be defined manually. Indeed, it is difficult to evaluate to what extent a score should have more weight than another one when computing the overall score. For instance, it is difficult to decide objectively if the exclusivity of an attribute in a class should prevail over the semantic proximity of this attribute with the other attributes of the class when calculating the global score ($s2$ vs $s3$ in the attribute recommender defined in Section 8.2). This problem is similar to the case where a software engineer would be asked to set the weights of each neuron in a neural network.

The use of machine learning is a solution to this weight definition issue. This mechanism allows the inference of the weights of each neuron from data labelled by users of the system. In the case of our software assistant, each of the four individual scores represents a neuron, connected to a single output neuron, which corresponds to the general confidence level of the system towards the recommendation.

The system is usually trained by minimizing a metric that measures the output error. This process is feasible when the output of the neural network can be determined and predicted in advance, to generate training data. In the modeling task, the goal of the assistant is not to provide the correct answer, but to provide useful and interesting ideas for the users. In order to be able to perform the learning process of the assistant, we therefore asked potential users of the system to prepare annotated data, which we exploited with machine learning. This operation is one of the main steps required to implement our modeling assistant. Each of these steps is explained in the following chapter.

Chapter 9

# Prototyping the software modeling assistant

*Chapter 8 provided the functionnal, structural, infracture, and requirements specifications for the construction of the modeling assistant prototype. In this chapter, we report on the technical implementation of the system as a prototype, based on the previous design artifacts. In each section, we detail the structure of each major component of the architecture.*

## 9.1 Architecture overview

Figure 9.1 presents the overall architecture of our system. It respects the 3-tier architecture defined in the framework, by implementing a user interface, the knowledge back-end, and the information repository. The *information repository* relies on a standalone Neo4j[1] graph database, which exposes a management interface. It is fed by the data collector, which is a Java SpringBoot[2] application, that gathers models from local and remote model repositories and integrates them in the knowledge base. The *knowledge back-end* implements the multi-criteria recommender system as a Java SpringBoot application. It communicates through Cypher queries with the Neo4j database that is hosted on the same server. The Java application features a learning mechanism that is able to exploit user-labeled data to infer the aggregation functions

[1] https://neo4j.com
[2] https://spring.io/projects/spring-boot

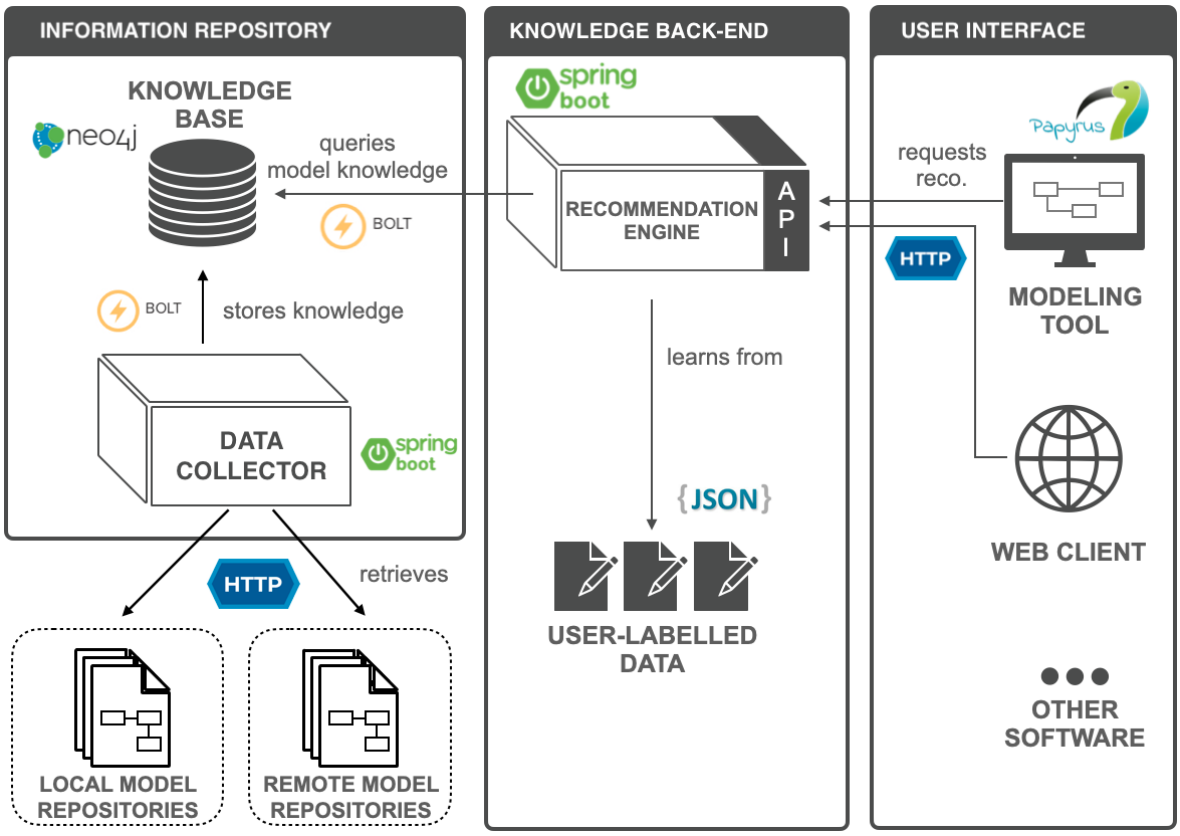



Figure 9.1 – The overall architecture of our prototype modeling assistant

weights with machine-learning. The *user interface* of the prototype consists of a Papyrus plugin that is embedded in the Eclipse environment. It communicates with the knowledge back-end through HTTP requests to retrieve recommendations to display.

This prototype is currently available online. It is deployed on a remote server, which hosts the Neo4j database and the knowledge back-end, executed in Docker[3] containers. A Continuous Integration/Continuous Delivery (CI/CD) pipeline is responsible for making the latest updates on the back-end available to the end-users of the assistant. The Papyrus plugin can be installed from the online update site[4]. The server is also available online, and access can be granted to the recommender system.

## 9.2 Building the knowledge base

This knowledge is created by the *data collector*, which embeds a *model processor*, whose role is to store the knowledge from the UML models of

[3]https://www.docker.com
[4]https://software-assistant.univ-lille.fr/modeling-assistant/composite/

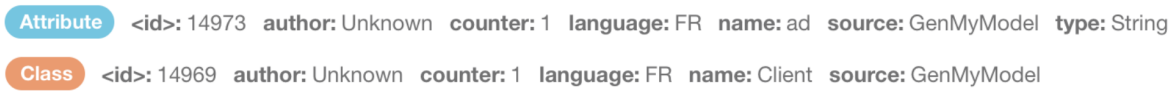



Figure 9.2 – Example of class and attribute nodes metadata in the graph database

*models repositories* into the knowledge base. It consists of a Java Spring-Boot application that converts xmi-structured models into subgraphs, and integrates them into the graph knowledge base. The models are analyzed when parsed and filtered or curated according to custom rules to avoid the use of toy models or the integration of misspelled names. The system also stores model meta-data in the subgraph such as the detected language of the model, its origin, or the type of each attribute to enable more precise filtering and explanations, as shown in Figure 9.2.

All subgraphs are stored in the knowledge base which relies on a standalone Neo4j graph database. It provides a web sandbox interface to perform manual queries, inspect the graph, and manage data, and a bolt API endpoint to expose a read/write to the *data collector* and the *knowledge back-end*. Knowledge retrieval is made through the use of Cypher queries, which enable to retrieve the data required to compute each individual score. An example of Cypher query is presented in Figure 9.3.

The instance contains knowledge extracted from around 116,000 UML models from GenMyModel public repositories. It represents more than 800,000 UML classes and more than 1,000,000 relationships between classes. The knowledge base server is hosted online on the same physical server as the recommendation engine, and is accessible with restrictions with a token granted upon request. A local knowledge base could be deployed on the private servers of a company to store sensitive knowledge from corporate domain-specific models. This domain-specific knowledge base could then complement our general base by receiving mirrored queries, and enable the system to provide more precise recommendations for specific business domains.

## 9.3 Implementing the knowledge back-end

This section details the behaviour of the application acting as the knowledge back-end. It reports on how recommendation score formulas are implemented, and provides an overview on how machine-learning is

**CYPHER QUERY**

```
$ MATCH (a:Attribute)--(c:Class {name: "Sensor"})
WHERE a.language = "EN" RETURN a, c
```

**QUERY RESULT**

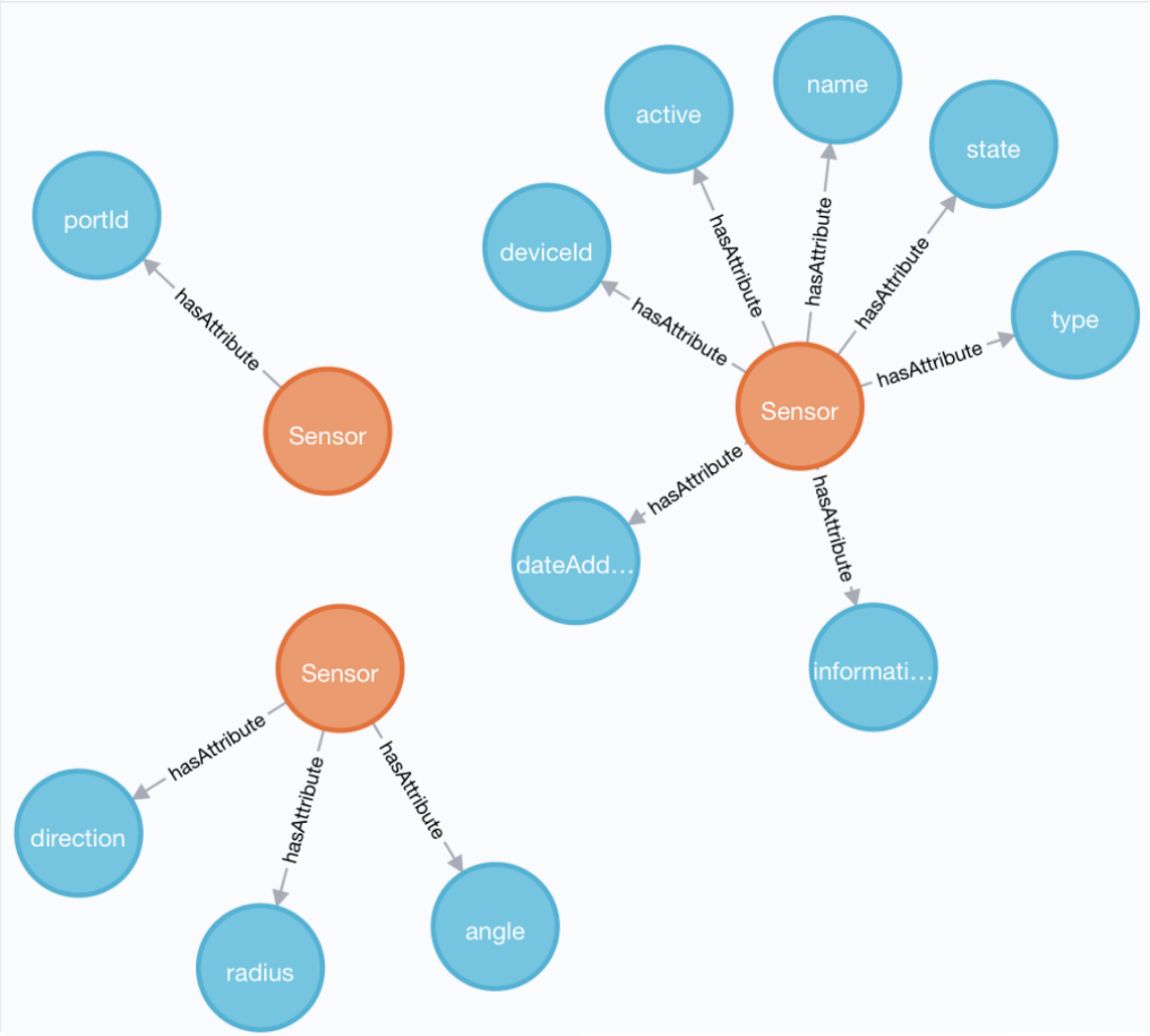



Figure 9.3 – Example of Cypher query

integrated to the modeling assistant.

### 9.3.1 Multi-criteria recommender system

The recommendations are produced and exposed through an API by the same Java SpringBoot application, that acts as the knowledge back-end.

This application implements the multi-criteria recommender system that is described in Section 8.2. Thus, it (i) communicates with the information repository, it (ii) computes attributes and classes individual recommendations scores, (iii) it aggregates individual scores into overall scores, and (iv) formats the results as a JSON file to be exploited by the clients. To retrieve the information from the knowledge base to compute the four scores, the application uses five Cypher queries for attributes recommendation, and five for classes recommendation. These queries are sent to the graph database using the Bolt protocol[5], operating over a TCP connection or a WebSocket. Each query is executed in a RxJava 2[6] flow, so that all queries can be executed in parallel. The results of these queries are transformed to Java objects by the Neo4j Object Graph Mapper[7] connector.

Each individual score is computed in the same RxJava flow than its related graph query to exploit multi-threading and reduce recommendation time. Scores are computed according to their descriptions in Section 8.2, and are linked to explanation elements related to the nature of the score. For instance, the individual score about context similarity will hold information about the classes that are shared between the source model and the currently edited model.

When all individual scores are available, the system analyzes the input request and determines the input context. This enables the system to select the correct aggregation function to compute the overall score. The aggregation function weights are determined according to each context, during the machine learning process that is described in Section 9.3.2. After calculating each overall score, recommendations are sorted in a descending order, from the highest overall score to the lowest. Only the 15 best recommendations are returned in the JSON response.

To produce recommendations, the system takes requests as input with the *target class name* (e.g., the class for which the user wants recommendations), the *current attributes names* of this class, and the name of the *other classes* in the model. The computed recommendations are

[5]https://boltprotocol.org

[6]http://reactivex.io

[7]https://neo4j.com/developer/neo4j-ogm/

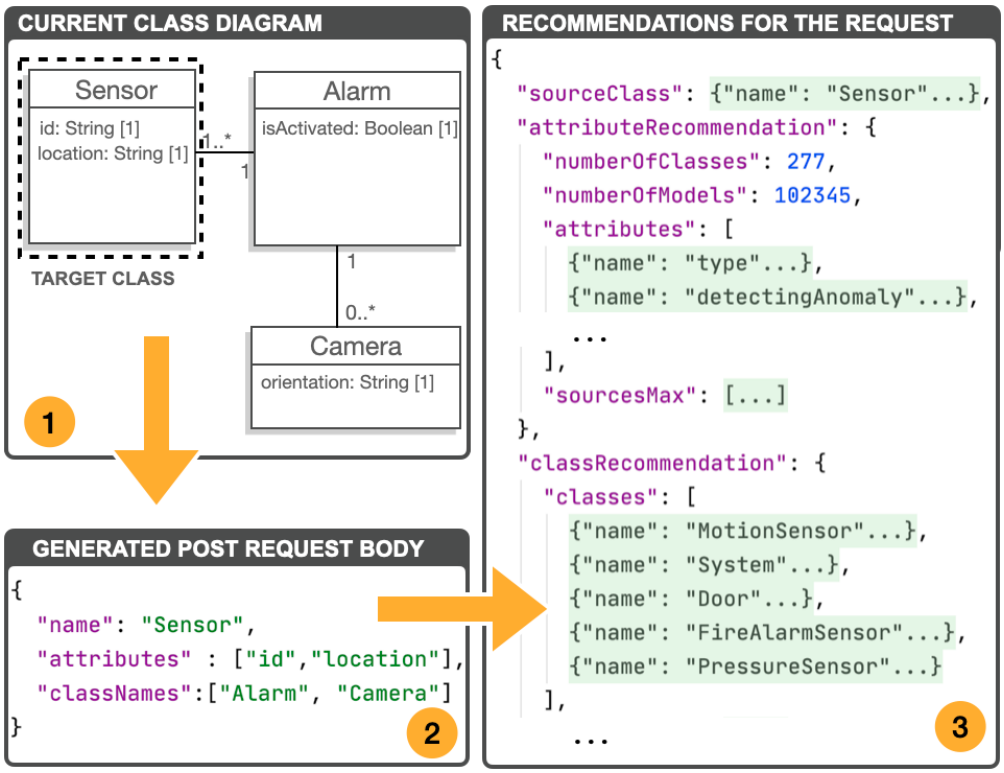



Figure 9.4 – Example of a recommender system query

returned in the JSON syntax, as presented in Figure 9.4.

### 9.3.2 Machine-learning mechanism

We implemented the machine learning mechanism as a linear optimisation feature. To infer the weights, we generate the different weight combinations for each context, with a predefined step. For instance, with a step of 0.1, each weight varies from 0 to 1, by take values 0, 0.1, 0.2, ..., 1. For each generated combination, we compute the $MTP@5$ precision metric such as described in Section 10.1.2. The target combination is the one that maximizes the precision metric.

This algorithm is implemented with Java, as a feature of the knowledge back-end. The data gathering details are provided in Chapter 10.

## 9.4 Integrating the assistant into Papyrus

As a means to provide users with recommendations, our modeling assistant implements a user interface. This interface consists of a Papyrus plugin, that is able to display in-editor suggestions. In this section,

we report on the implementation of the plugin, and describe the user interface.

### 9.4.1 Creating the Papyrus assistant plugin

For the recommendations to be directly exploited by software engineers, they must be presented in a user interface. As part of our solution, we propose a plugin integrated to the Papyrus modeling tool that we call our *Papyrus client*. As it is fully integrated into this modeling tool, our client can directly add the chosen recommendations, such as new classes, attributes, or relationships into the model which is instantly updated in the editor. As the update is bidirectional, manual changes in the model from the editor trigger the update of the recommendations.

The client relies on the plugin mechanism for the Eclipse framework, is coded in Java, and is available as an update site for a user to install it in their own Papyrus environment. When a user clicks a class in the editor, the plugin retrieves the edited model as well as the selected class to form a recommendation query as shown in Fig. 9.4 (1) and (2). The query is sent to the recommendation engine which returns a response JSON object as depicted in (3).

The user interface is displayed as an overlay panel when triggered the key binding ⇧+⟵. It can be decomposed into 3 distinct parts: the *class overview*, the *recommendation panel*, and the *explanation area*.

### 9.4.2 Designing the user interface

#### The class overview

The user interface of the assistant features a clone of the class for which recommendations are provided. It enables users to stay aware of the current state of the class, while allowing the system to draw visual indicators within the class representation. Figure 9.5 (1) shows an overview for the class `Square`, which owns one attribute `width`, with multiplicity 1, `public` visibility and type `Integer`.

The first visual indicator is a pie chart located in the upper-right corner of the class. It represents the value of the score S1 (see Section 8.2) for the attribute hovered in the *recommendation panel*. The second visual indicator consists of horizontal bars located under each attribute of the recommendation overview. It describes the strength of the synergy between the attribute hovered in the recommendation panel and each attribute own by the currently edited class.

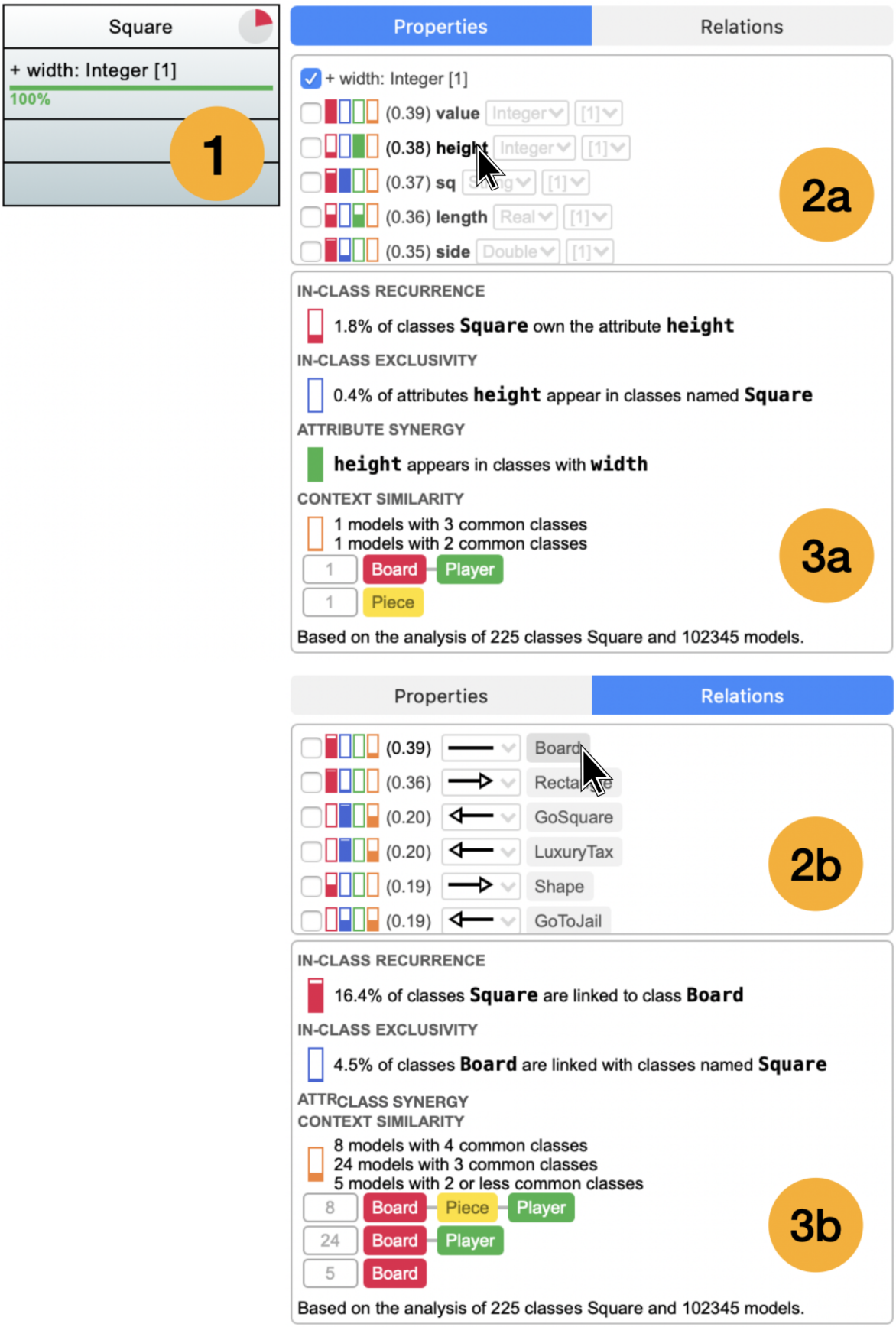



Figure 9.5 – The interface of the modeling assistant.

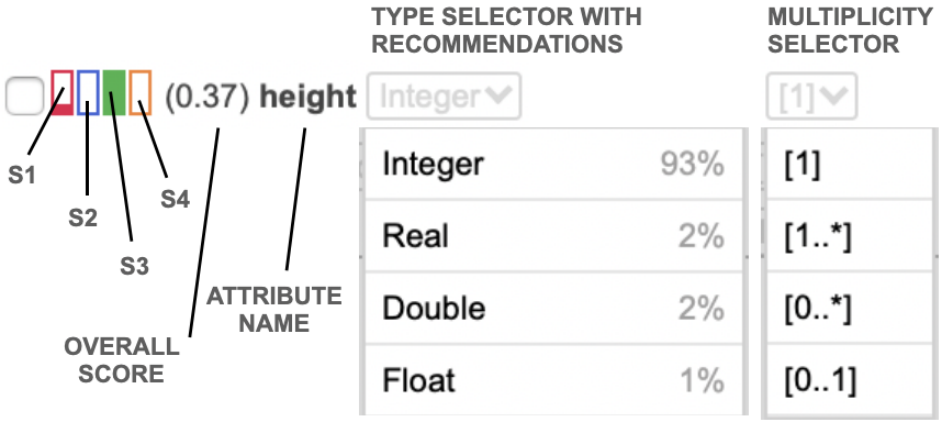



Figure 9.6 – The anatomy of an attribute recommendation entry.

### The recommendation panel

Figure 9.5 (2a) and (2b) respectively show the attributes and the relationships recommendation panels. These two views are accessible by choosing the corresponding tab, and both present recommendation elements as a scrollable and clickable list. A click on the checkbox of an entry triggers the addition or the deletion of this element in the model.

Each line of the panel shares the same anatomy, as described for an attribute in Figure 9.6. The checkbox triggers model update, the four vertical progress bars represents the normalized values of each of the individual scores described in Section 8.2. The following numerical value is the overall confidence score, based on the aggregation of the individual scores. The default type of the attribute is the most recommended one. This can be edited by clicking the type label and selecting another type from those provided in the selector menu, ranked according to the recommendation data. Multiplicity is set to 1 by default and can be edited as well, but no recommendation is provided. Relationship entries only differ by the relationship type selector, which contains recommendations for relation type similarly to attribute types (see Figure 9.5 (2a) and (2b)). Attribute visibility, and relationship name and multiplicities are not yet supported by our modeling assistant.

### The explanation area

The user interface features an explanation area which provides detailed information about the currently hovered entry. This information (as displayed in Figure 9.5 (3a) (3b)) includes an explanation for each of the individual score in the current context, and the number of analyzed classes and models. The purpose of these explanations is (i) to enable

users to increase their knowledge due to the experience acquired by the system through the analyzed models, and (ii) to better understand the inner mechanisms of our recommender system. This is a step towards improving system transparency to increase its trustworthiness and acceptability [73].

## An early validation of the formal framework

Validating conceptual frameworks is not an obvious task. In their study, Meseguer [113] defined validation as a global term encompassing both verification and evaluation. Verification refers to completely examining the system against its specifications, while validation consists in examining whether a particular design meet its intended purpose and perform as expected. In their book, Jabareen [82] mentions the process of validating the conceptual framework as follows "*the aim in this phase is to validate the conceptual framework. The question is whether the proposed framework and its concepts make sense not only to the researcher but also to other scholars and practitioners. Does the framework present a reasonable theory for scholars studying the phenomenon from different disciplines? Validating a theoretical framework is a process that starts with the researcher, who then seeks validation among outsiders*".

Two common procedures for validating conceptual frameworks emerge from the existing literature: *comparison* or *instantiation*. Comparison relies on the availability of other frameworks with which to compare the proposed framework. Then, an analysis can be performed to highlight benefits and issues with the different considered approaches. Instantiation refers to exploiting case studies to design systems according to the studied framework, and demonstrating that the created system meets the expectations. The strength of this validation approach hence depends on the selected case studies.

As a first validation step, we demonstrated that the formal framework defined in Chapter 7 successfully enabled the design and the implementation of an instance of a modeling assistant. As a step further towards validation, the next chapter introduces the evaluation of the recommendation algorithm and two case studies that highlight the accuracy of the system.

# Chapter 10

# Early evaluation of our system

*Chapter 8 and 9 detailed the design and the implementation of a prototype modeling assistant, based on the formal framework defined in Chapter 7. To conclude on the performance, potential benefits, or influence of this prototype, empirical evaluations must be conducted. This thesis does not present the results of such user experiments, that are planned to be conducted as future work. Nevertheless, this chapter introduces the evaluation of the recommender system based on user-labelled training and testing datasets, and introduces two case studies to demonstrate the behaviour of the system.*

## 10.1 Evaluation of the modeling recommender system

The preliminary evaluation of our approach is based of assessing improvements in the quality of the recommendations, as well as the adequacy of system control, information transparency, and system transparency. To the best of our knowledge, no similar approach can be found in the literature. A replication package containing the labelled data, the original files and the metrics source code is available online[1]. The evaluation involved data from over 95,000 models, different from the data set used for the implementation of the system in the previous

[1] https://hufamo.univ-lille.fr/modeling-assistant


Figure 10.1 – The web data-labelling interface

chapter. This new data set contained approximately 634,000 classes and 616,000 attributes. The models were also retrieved from the GenMyModel[2] public repositories by courtesy of Axellience. For quality purposes, we only selected models greater than a minimum size (over 10 kilobytes).

### 10.1.1 Data gathering

We collected labelled data through a dedicated web interface during a preference-elicitation phase. This interface, as presented in Figure 10.1, first presents multiple situations where *class diagrams with recommendation target class* are presented one at a time. A list of unranked candidates —potential recommendations— is displayed for each situation. Using this interface, the user is asked to remove all attributes that do not fit semantically in the presented situation. Once this is completed, the user is then asked to create a ranked list of the top 10 best recommendations from the displayed elements. This task should be repeated for multiple situations in different contexts a sufficient number of times in order to collect enough information to determine the four utility functions. Once this data is collected, it is used to calculate $a_k$, $b_k$, $c_k$, $d_k$ weights from the aggregation functions in such a way that they maximize the Mean Top Average Precision metric defined in Section 10.1.2.

[2]https://www.genmymodel.com

We gathered 9,858 labelled attributes from 30 participants: 9 senior and 4 junior researchers, 3 senior and 12 junior developers from industry, and 2 M.Sc. students. Prior to starting the labelling exercise, participants were asked to answer questions about their familiarity with UML and the extent of their modeling work. On average, participants estimated their knowledge of UML class diagrams to range between fair and good (mean: 3.5 on 5-point Likert scale, std. deviation: 1.0). This assured us that participants had a relatively good understanding of the context and consequently, and that the information gathered was semantically meaningful. Participants were asked to respond to up to 20 examples of different situations: 5 per context. In order to minimize the impact of participant fatigue on the results, the 20 examples were randomly displayed and participants were allowed to respond in several sessions.

### 10.1.2 Evaluation metrics

To more accurately evaluate the attribute recommendations, we compared the ranked results of our system with the ranked preferences as chosen by the users who created their top-5 ranked list. Consequently, we computed metrics for just the top-5 recommended attributes; i.e., the five attributes with the highest scores.

**Precision@5** (P@5) is the proportion of recommended items that a user deemed as belonging in the top-5 list of relevant attributes. In our case, relevant attributes were those that were not excluded by the user from the candidate list.

$$P@5(\text{set}) = \frac{\text{n° of relevant items in system top-5}}{5} \tag{10.1}$$

**TopPrecision@5** is the proportion of recommended items in the top-5 list provided by the recommender system that are also included in the top-5 set chosen by the user.

$$TP@5(\text{set}) = \frac{\text{n° of common items in user and system top-5}}{5} \tag{10.2}$$

**TopAveragePrecision@5** (TAP@5) takes ranking into consideration in evaluating the mean average precision of the top-5 of the system. Mean Average Precision (MAP) is a popular metric for measuring recommendation algorithms in information retrieval. We defined TAP@5 as follows:

$$TAP@5(\text{set}) = \sum_{n=1}^{5} \frac{P(n) \times pos(n)}{R} \tag{10.3}$$

Table 10.1 – Labelled data distribution

| | General | Ctx. 1 | Ctx. 2 | Ctx. 3 | Ctx. 4 |
|---|---|---|---|---|---|
| Training | 8,146 (205 sets) | 2,462 (50 sets) | 2,338 (43 sets) | 1,765 (59 sets) | 1,581 (53 sets) |
| Testing | 1,712 (40 sets) | 544 (11 sets) | 562 (10 sets) | 303 (9 sets) | 303 (10 sets) |
| Total | 9,858 (245 sets) | 3,009 (61 sets) | 2,900 (53 sets) | 2,068 (68 sets) | 1,884 (63 sets) |

where pos(k) indicates whether the element from system top-5 in position *k matches* the position of the element in a user's top-5 list, while R refers to the number of elements for which $pos(k) = 1$; P(k) is the ratio of correctly recommended elements over top-k recommended elements.

These metrics can be computed for each ranked set of attributes. Therefore, as users provided several sets of attributes, we considered the means of these metrics as follows:

$$Mm(S) = \sum_{s \in S} \frac{m(s)}{N} \tag{10.4}$$

where $m$ is the metric for which the mean is calculated (i.e., MP, MTP, and MTAP); $S$ is the data set for which the mean was computed, and $N$ is the number of elements in $S$.

### 10.1.3 Metrics results

We obtained 245 sets of labelled data from users, which constitute a corpus of 9,858 attributes distributed for training and testing phases, as presented in Table 10.1. We trained our overall rating functions with 81% (205 sets) of the total labelled data set and obtained the functions presented in Table 10.2. The goal of the training was set to the maximization of the MTP@5 metric, as it is the most representative possible improvement of our system when compared to unassisted user selections.

The added value of using machine-learning is demonstrated by the evolution of the metrics before and after the machine-learning process. Both situations only differ in the the values and distribution of weights in the utility functions. We set up the initial configuration (i.e., before ML) by setting the weight values to be equal. For instance, we define the initial aggregation function for Context 4 as $overall_4 = 0.25 \times s1 + 0.25 \times s2 + 0.25 \times s3 + 0.25 \times s4$. The final configuration corresponds to

Table 10.2 – Learned overall functions

| Context | Overall Function |
|---|---|
| Context 1 | $0.80 \times s1 + 0.20 \times s2$ |
| Context 2 | $0.03 \times s1 + 0.01 \times s2 + 0.96 \times s3$ |
| Context 3 | $0.51 \times s1 + 0.03 \times s2 + 0.46 \times s4$ |
| Context 4 | $0.56 \times s1 + 0.03 \times s2 + 0.29 \times s3 + 0.12 \times s4$ |

Table 10.3 – Testing data set metrics measures

| Metric | General | Ctx. 1 | Ctx. 2 | Ctx. 3 | Ctx. 4 |
|---|---|---|---|---|---|
| Initial MP@5 | 87.0% | 83.6% | 90.0% | 84.4% | 90.0% |
| Initial MTP@5 | 34.7% | 23.6% | 38.7% | 36.7% | 40.0% |
| Initial MTAP@5 | 27.8% | 4.5% | 22.5% | 11.1% | 35.0% |
| Final MP@5 | 91.5% (+4.5%) | 92.7% (+9.1%) | 90.0% (-) | 93.3% (+8.9%) | 90.0% (-) |
| Final MTP@5 | 51.0% (+16.3%) | 56.4% (+32.7%) | 49.3% (+10.7%) | 52.2% (+15.5%) | 46.0% (+6.0%) |
| Final MTAP@5 | 42.5% (+14.7%) | 50.0% (+45.5%) | 35.0% (+12.5%) | 50.8% (+39.7%) | 45.0% (+10.0%) |

the application of the utility functions defined in Table 10.2.

We evaluated the overall utility functions on a specific testing data set, which represents 19% of the full labelled data set, (see Table 10.1). The other 81% were used for training purposes. The results of the metrics evaluation are presented in Table 10.3.

### 10.1.4 Discussion

The initial general MP@5 is pretty high (87.0%). The low impact of the learning process on this score (+4.5%) indicates that the different criteria already strongly converge to recommend user-relevant attributes, and that the impact of the weights on the overall rating functions are, in that case, secondary. The impact of the supervised-learning process becomes more important according to the desired quality of the recommendations. Indeed, the initial low value of MTP@5 increases from 34.7% to 51.0% after the learning process. This means that, on average, more than two attributes of the 5 first recommendations of the system are attributes that participants included in their top-5 best recommendations. Following the learning step, MTP@5 shows the most significant increase among all metrics (+16.3%). The utility functions were defied so as to optimize this metric. MTAP@5 takes differences in recommendations ranking between

system and user top-5 into account. Only high-quality recommendations increase this metric, which explains why it has the lowest initial values for all contexts. With a final value of 42.5%, MTAP@5 indicates that, on average, more than 2 attributes of the 5 first recommendations are in participants' top-5, likely to be in top positions and ordered as the participants expected.

The empirical results obtained indicate that our approach provides acceptable results (on average, more than 4 recommended attributes out of 5 are deemed relevant, and also 2 recommendations out of 5 appear in users' top-5 rankings). However, as pointed out earlier, it is too early to make any firm conclusions about the effectiveness of our approach compared to alternatives until further evaluations are performed. In addition, we can draw the following conclusions from the evaluation:

- The initial effectiveness measure that we proposed here looks as if it could serve as a common metric for future related work.
- The defined criteria do seem to reflect information trustworthiness. Moreover, the rationale behind them can be easily explained, which means that they do support information transparency.
- The linear utility function approach we used allows any overall score to be traced to each criterion used to derive it. This enables users to understand the inner mechanisms of the system and thus supports system transparency.
- The utility function can either be set manually or defined using supervised-learning. These settings allow users to have control over the results that are presented giving them control of the system.

## 10.2 Use cases

In this section, we introduce two use cases in which we demonstrate the behaviour of the system. For both scenarios, let us say that this expert exactly knows what she wants to model, and then uses the modeling tool to digitalize the model into one class diagram. We assume that she uses the Papyrus modeling tool with our modeling assistant. To mimic the constraints on model elements to create, we use the class diagram in Figure 10.2 as the diagram that should be replicated. Fixed names and relationships act as the modeling constraints the architect wants to respect. This diagram is extracted from [93], one of the best-seller UML course books.

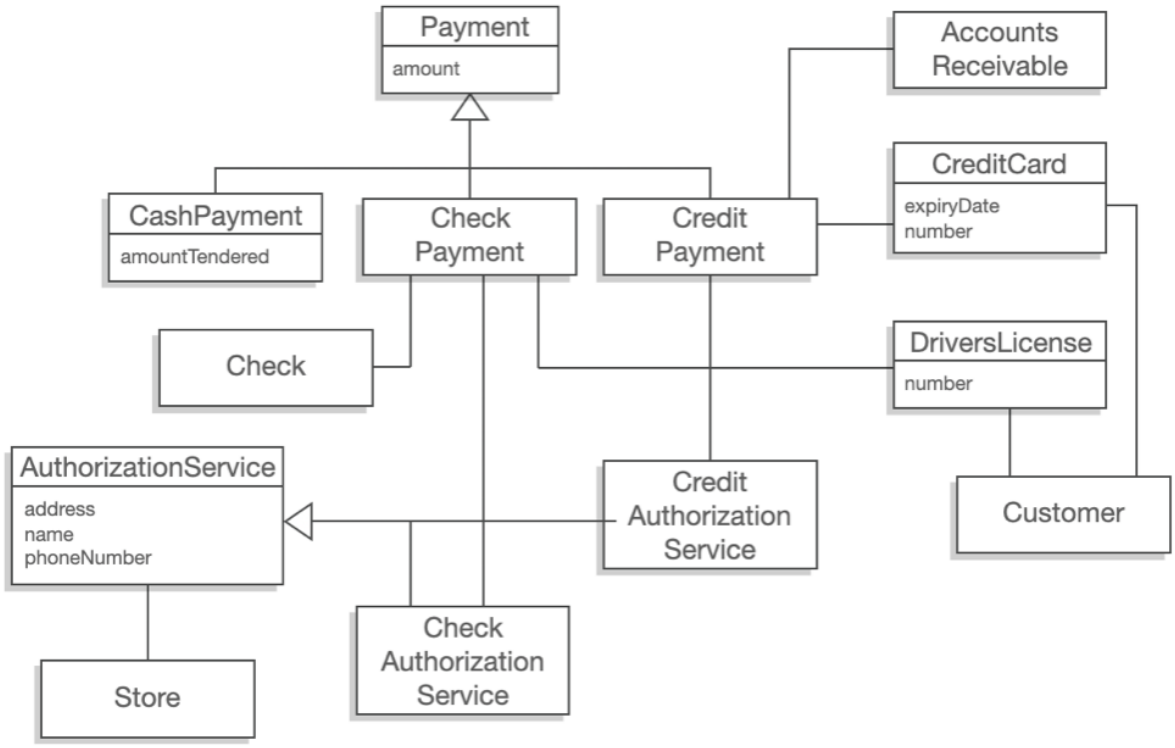



Figure 10.2 – The diagram to replicate in the use cases.

### 10.2.1 Case 1: the general knowledge base

In this first use case, we make the assumption that the user has never modeled the domain concepts she has to represent. Therefore, no model concerning details about the business logic is integrated in the general knowledge base that the recommender system exploits. To build the diagram, we first created the class, tried to add its attributes based on the recommendations, and then tried to add relationships based on the recommendations. If no suggestion was correct, we created the missing element manually, and repeated the same pattern until the diagram was complete. Note that when accepting a relationship recommendation, the system automatically creates the class if it is not yet in the diagram.

Out of the 35 model elements to replicate, the system was able to recommend 10 elements correctly (29%), 3 elements with the same sense but not the exact same name (8%), and could not recommend the 22 other elements (63%). This distribution is represented in Fig. 10.3 (a).

### 10.2.2 Case 2: adding domain-specific knowledge

This case presents the behaviour of the system when the knowledge base includes the domain concepts that the architect wants to model. To simulate this situation, we created the diagram of Figure 10.2 as an xmi

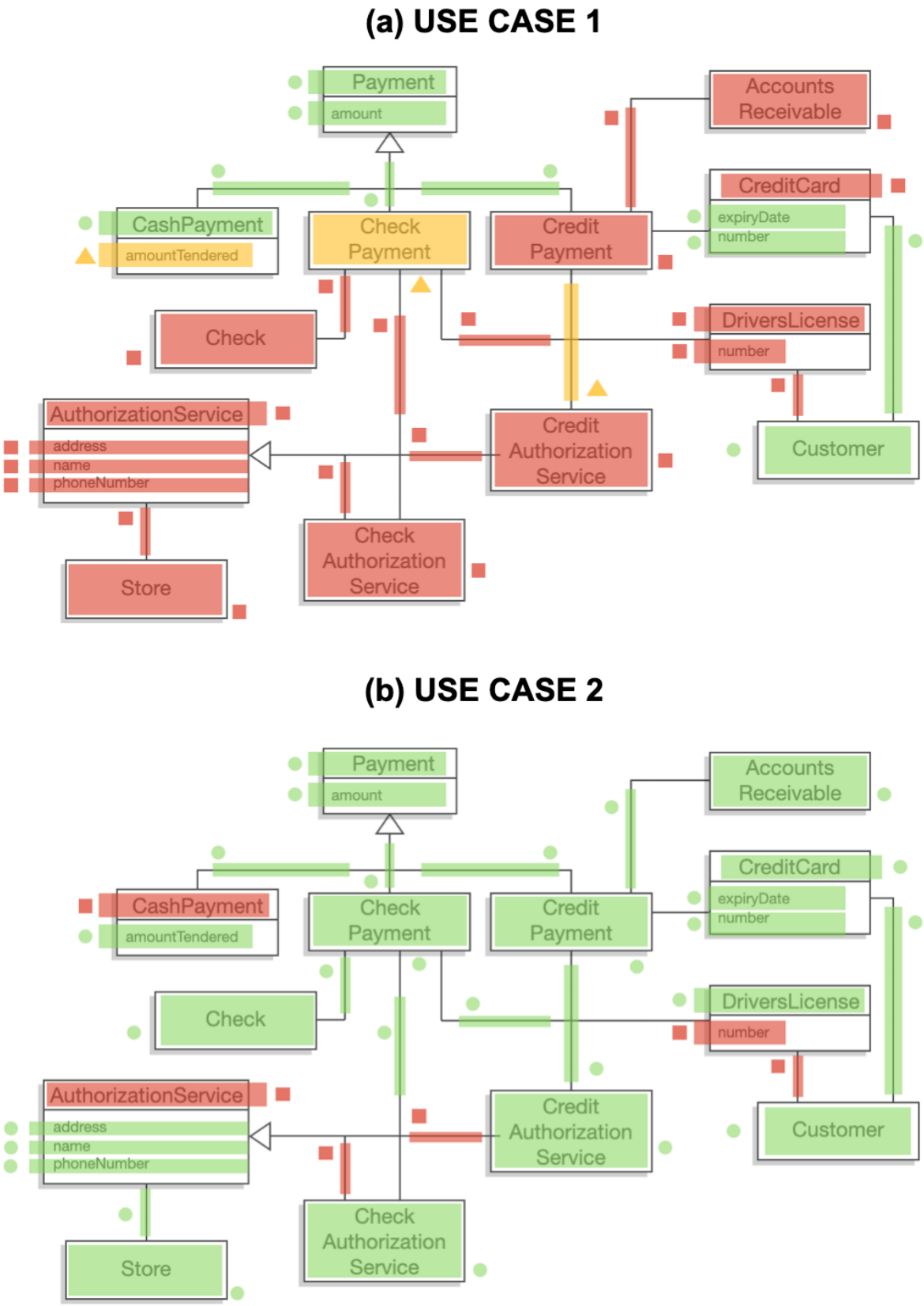



Figure 10.3 – Exact recommendations (green/circle), approximate recommendations (orange/triangle), and not recommended elements (red/square) in both use cases.

file, store it in a local folder, and run the model processor (see Figure 9.1) to integrate it among the 100,000+ models of the knowledge base. This illustrates the behaviour of a system featuring a company-specific model repository. Then, we reproduced the model creation steps of the first use case.

Out of the 35 model elements to replicate, the system was now able to recommend 29 elements correctly (83%), and could not recommend the 6 other elements (17%). This distribution is represented in Fig. 10.3 (b).

### 10.2.3 Discussion

The efficiency of the system depends on the set of models on which it is based. As a starting point, we provide a general knowledge base enabling the assistant to support the modeling of general concepts. The system can then be extended with domain-specific models to learn about business logic and provide recommendations more suitable for industry, as illustrated in the case studies. Feeding the system with one new class diagram led to increase the ratio of correct recommendations by 54%. Moreover, even wrong or approximate recommendations could help beginners and experts strengthen the completeness of their model, by suggesting new or forgotten directions to explore. As discussed in [157], this recommendation approach might be generalizable to other types of diagrams.

## An early evaluation of the system

This chapter presented two directions for evaluating the prototype proposed in Chapters 8 and 9. It first presented an evaluation of the core recommender system, by assessing its suggestions over user-labelled data. The results of this study demonstrated that the defined criteria seem to enable the computation of accurate recommendations. The aggregation method also enables information-trustworthiness to be expressed because of to the linear combination of individual scores. The case studies show that the general knowledge base of the prototype allows for modeling recommendations across multiple domains, with a lower recommendation rate. It emphasizes the need for companies to make their internal models available to the system, so it can unleash its recommendation power.

These results seem to indicate that the prototype might empower users with modeling recommendation and hence boost their creativity

or productivity, while reducing the time and effort required to produce models. However, such results are still to prove with proper empirical evaluations with users. These experiments are planned to be conducted as future work, as described in the conclusion of the thesis.

# Conclusion

In this section, we conclude on the results presented in the thesis. We first summarize the research approach that has been followed, then we present the main results echoing the thesis research questions, and finally we identify new research lines for future work.

## Research approach rationale

The purpose of this thesis was to study the possibility of augmenting software modeling engineers with software assistants. To this end, we sought to identify aspects of modeling that could be assisted, and investigated how these problems could be assisted. In our research approach, we first tried to define software assistants for modeling, in order to study their ability to address modeling problems. We then defined the nature of the modeling task and identified the problems related to it that are evoked in the literature. This work highlighted the complexity of assisting the modeling task, due to its creative and ill-structured nature, as well as the limited scientific resources available on the foundations of software modeling assistance. To better understand how this assistance was realized in the research landscape, we also conducted a systematic mapping study on software assistants in software engineering. This study revealed the low presence of implemented and usable assistants, capable of helping in modeling. Thus, to better understand how to build useful software assistants, both in terms of functionality and the way they help, we conducted a series of interviews with modeling experts. These discussions have allowed us to outline ways to support modeling for beginners as well as for experts, and to provide indications on how to interact with them. Due to to the exploratory work of this first part, we have been able to draw a big picture of modeling assistance in general, and to identify the four main notions on which software assistance to modeling is based.

The second part of the thesis relies on the conclusions of this big picture, to formalize the design of software assistance for modeling. We first identified the resources in the literature that address the key notions of modeling from the big picture, concerning Trust, Creativity, Recommendations, and Automation. The theoretical papers and frameworks gathered allowed us to build a series of constraints to the design of computer systems capable of providing modeling assistance, with respect to these 4 key notions. From these constraints, we then defined a conceptual framework according to the ISO 42010 standard aiming at guiding the design of software assistants for modeling. This framework provides four architecture viewpoints, allowing to describe the architecture and the specifications of the system to be designed.

In the third part of the thesis, we have evaluated the validity of this framework, by proposing a first validation effort in two steps. First, we described the specifications of the system from the framework defined in the second part, and technically implement the prototype assistant. The realization of this prototype of assistant first tested the capacity of the framework to be used to define functional and usable systems. In a second step, we exploited this prototype to conduct preliminary performance evaluation experiments. Although they are only indicative and require further empirical work with users, these experiments indicated the performance of the prototype in providing recommendations deemed accurate by users.

## Main results

Our results highlight the need for modeling engineers using modeling tools to benefit from assistance. At the same time, we reveal a gap in the literature regarding software assistance for modeling. Our work aims to fill this gap, paving the way for the study and design of more modeling assistants, both in research and in industry. The results of the thesis allow to identify the main functionalities of modeling assistants to be investigated as research lines or implemented as prototypes. In order to realize these systems, we propose a framework for the design of software assistants for modeling, allowing to guide the creation of such systems. Finally, we propose a modeling assistant to help the design of UML class diagrams by recommending attributes and relationships for UML classes.

## Research questions

In this section, we present how the results presented in this manuscript address the thesis research questions.

**T.R.Q. 1. What is a software assistant for software modeling?** A software assistant for software modeling is a software bot that provides users with valuable knowledge to help them identify, understand, or solve a modeling problem. Modeling assistants may help users make a decision and eventually perform a task according to this decision with a certain degree of autonomy because of this knowledge. Their outcomes might not be deterministic, as they adapt to each problem and context. They might be used to automate manual tasks to save time and reduce effort, but they must be able to come up with new information and ideas and that may be valuable to increase the knowledge of the user. Modeling assistants consist in complete and ready-to-use software systems, accessible through a user interface (to be able to provides users with valuable knowledge).

**T.R.Q. 2. Are there software assistants for software engineering (and software modeling) in the literature?** Our study of the literature, through a systematic mapping study, identified 47 software assistants for software design, software construction, software maintenance, and cross-cutting socio-cultural systems such as networking tools. Among them, only 2 out of 47 systems were aimed at supporting software modeling. This demonstrates the poor effort of the scientific literature to develop modeling assistants prototypes, that are required to conduct empirical experiments with users.

**T.R.Q. 3. What are the common characteristics of software assistants for software engineering from the literature?** According to the results of our systematic mapping study, software assistants for software engineering could be classified in three main groups *informers*, *passive recommenders*, and *active recommenders* based on their skills. These systems mainly appear to not allow users to provide feedback on the potential recommendations they generate, to not provide explanations with their suggestions, and to rarely indicate how much confident they are with their suggestions. Software assistants for software engineering tend to feature one of five general automation patterns about their interactions. It appears that they never make decisions fully autonomously and require the user to take part in the decision selection process.

**T.R.Q. 4. Are there identifiable ways of designing software assistants for software engineering that emerge from the literature?** To the extent of our knowledge, no previous research work intended to provide guidelines for designing software assistants for software engi-

neering. No specific correlation emerged during the result analysis of our systematic mapping study, which tends to indicate that no specific design approach could have been identified. Moreover, the previous research question highlighted that aspects such as the design of interactions, with the handling of Human-Computer Interactions (HCI) indicators are rarely considered when conceiving the assistants identified in the literature. This highlights that, while there seem to exist no framework to design software assistants for software engineering, these systems are designed with a poor consideration for aspects related to human factors.

**T.R.Q. 5. Are the software assistants available in the literature in line with the expectations of modeling practitioners?** Our interviews with experts enabled us to collect modeling engineers' needs and expectation about modeling assistance. The results of this work led to the identification of 21 potential assistance features to support engineers in their work. Practitioners emphasized the need for proper interaction and automation design, and highlighted the need for the assistants to implement a confidence indicator system, an explanation interface, a user-feedback collector. This clearly breaks with the observed state of the software assistant literature, as discussed in T.R.Q. 3 and 4. The poor availability of modeling assistants also clashes with the engineers' stated need for assistance. Thus, research appears to not be ready for providing modeling assistants to practitioners, or support their design. If, by chance an modeling assistant can be exploited from the literature, it is likely that it will not match practitioners' expectations about interactions and automation.

**T.R.Q. 6. What are the key concepts in modeling assistance?** Our work expresses an original vision of modeling assistance, which is based on *trust*, *creativity*, *recommendations*, and *automation* notions that are already anchored in the software engineering landscape. Among the four key notions of modeling assistance, we consider the notions of *recommendations* or *automation* which are common subjects in the scientific computer science community, and which are even research fields in their own right. The notion of *trust* is recently emerging in computer science, especially due to trust issues with artificial intelligence. These problems are at the origin of the emergence of research domains such as the AI explainability, or the creation of consortia of academics and industrialists aiming at studying this issue. Finally, we deal with the *creativity* issue which is rarely treated in the computer science community. This thesis has brought to light elements of the literature on creativity from psychology, applicable to computer science tasks. The creative nature of

the modeling task conditions the design of software systems aiming to integrate it. We hope that our work will help to address the skepticism of the computer science community towards creativity, and especially the doubts expressed about the ability of a software assistant to support creativity. These notions of trust and creativity, both *human* characteristics, justify the necessity and the soundness of our user-centred research approach.

**T.R.Q. 7. What are the guidelines to follow when designing software assistants for software modeling?** The second part of this thesis aimed at providing a conceptual framework to design software assistants for software modeling. The framework is defined in Chapter 7 according to the ISO 42010 standard. We define four architecture viewpoints about the *functional architecture*, the *structural architecture*, the required *infrastructure* for the system, and a set of 22 functional *system requirements* to design the assistant and its features. As a tutorial for using the framework, these guidelines are applied to the design of prototype modeling assistant in the third section of the thesis.

# Future work

In this section, we identify directions to extend the work presented in this thesis. We first mention the future directions related to the prototype that has been presented, and then list the potential research lines related to our framework for software modeling assistants.

As we mentioned in Chapter 10, our modeling assistant prototype calls for further evaluation and validation. We are particularly interested on assessing the impact of the system on users' productivity, and creativity. Measuring these two user-related characteristics requires the prototype to be tested during a controlled empirical experiment. Productivity may for instance be assessed through a read-and-reproduce experiment, where participants are provided with existing diagrams that they are requested to reproduce. The time required to accomplish the task can then be measured for participants with and without the assistant. Creativity may be assessed through an idea-generation experiment, where participants are required to produce the maximum number of different versions of the same system. The experiment then should in turn be conducted with and without the assistant, to compare the number of versions produced with and without assistance.

As the system should foster the trust relationship, we also aim at evaluating how users perceive the trustworthiness of the system. Assessing trust remains a complex topic. However, several research efforts

recommended the use of surveys and polls to assess users' opinion and perception about the trust relation they established with the system.

The prototype assistant is freely available, deployed on an online server, and can be downloaded from our update site to be integrated to Papyrus. We plan to maintain the prototype and integrate more knowledge into the knowledge base. This first could be achieved by adding more external repositories to the data collector. We have already conducted experiments aiming at collecting xmi files representing models (i) from public repositories on GitHub[3], and (ii) from computer-generated images of diagrams obtained by searching on Google. These projects have shown satisfactory results and are to be integrated into the modeling assistant prototype. The second option to collect more knowledge is to implement a mechanism to store the recommendation choices made by the user. This would enable us to build diagrams from users' choices, that could be included in the knowledge base as a local data repository.

With more users, we could refine the aggregation functions and thus refine its behaviour. New criteria could be added to the system, to take advantage of the metadata of our knowledge, such as the author, the model data, or its source.

Besides the prototype-related directions, another research line focuses more on the HCI aspect, and relates to the evaluation of the explanation interface. It consists in investigating if the color indicators, the textual rationales, the icons, and the interactions are well understood, and adapted to the users. This is inline with our user-centred research approach to design systems tailored to the users' needs, and would also lead to a deeper understanding of the modeling tool users' cognitive processes.

This manuscript presented research conducted by interviewing 16 modeling practitioners, as a first qualitative effort to better understand their need for modeling assistance. As another research line, we aim to strengthen the outcomes of this study by conducting further quantitative research. We plan to publish an online survey, built on the results of our qualitative results, to collect more data and thus assess the generalizability of the hypotheses we formulated.

One of our research directions is to strengthen the validation of our framework. To do so, we plan to exploit it to define new software assistants that might help with the design of other diagrams, with UML or other languages. Indeed, the knowledge in the knowledge base in language-independent, and thus can be applied to a broad variety of cases. For instance, we could define modeling assistants supporting

[3] https://github.com

SysML, or completely informal modeling.

We also aim to investigate the use of software assistants during other phases of software design. Our interviews with practitioners highlighted the need for assistance in phases other than model edition. This includes brainstorming sessions, or meeting with clients. These phases share the creativity and problem-solving characteristics of the modeling task, and hence might qualify for the integration of software assistants similar to modeling assistants, which could integrate knowledge during their tasks.

# Bibliography


[1] Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, Lionel C. Briand, and Eduardo Vaz. "A machine learning-based approach for demarcating requirements in textual specifications". In: *IEEE 27th International Requirements Engineering Conference (RE)*. 2019, pp. 51–62.

[2] Phillip L. Ackerman and Margaret Beier. "Knowledge and intelligence". In: *Handbook of understanding and measuring intelligence*. 2005, pp. 125–139.

[3] Russell L. Ackoff. "From data to wisdom". In: *Journal of applied systems analysis* 16 (1989), pp. 3–9.

[4] Gediminas Adomavicius and YoungOk Kwon. "Multi-criteria recommender systems". In: *Recommender Systems Handbook*. 2015, pp. 847–880.

[5] Gediminas Adomavicius and YoungOk Kwon. "New recommendation techniques for multicriteria rating systems". In: *IEEE Intelligent Systems* 22 (2007), pp. 48–55.

[6] Gene M. Alarcon, August Capiola, and Marc D. Pfahler. "Chapter 7 - The role of human personality on trust in human-robot interaction". In: *Trust in Human-Robot Interaction*. 2021, pp. 159–178.

[7] Lissette Almonte, Esther Guerra, Iván Cantador, and Juan de Lara. "Recommender systems in model-driven engineering". In: *Software and Systems Modeling* (2021).

[8] Muhammad Asaduzzaman, Chanchal K. Roy, Kevin A. Schneider, and Daqing Hou. "FEMIR: A tool for recommending framework extension examples". In: *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2017, pp. 967–972.

[9] Colin Atkinson and Thomas Kühne. "Reducing accidental complexity in domain models". In: *Software and Systems Modeling* 7 (2008), pp. 345–359.

[10] Alberto Bacchelli, Luca Ponzanelli, and Michele Lanza. "Harnessing Stack Overflow for the IDE". In: *3rd International Workshop on Recommendation Systems for Software Engineering (RSSE)*. 2012, pp. 26–30.

[11] Reinhard Bachmann and Akbar Zaheer. *Handbook Of trust research*. 2006.

[12] Omar Badreddin, Rahad Khandoker, Andrew Forward, Omar Masmali, and Timothy C. Lethbridge. "A decade of software design and modeling: A survey to uncover trends of the practice". In: *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2018, pp. 245–255.

[13] Michael D. Baer, Fadel K. Matta, Ji Koung Kim, David T. Welsh, and Niharika Garud. "It's not you, it's them: Social influences on trust propensity and trust dynamics". In: *Personnel Psychology* 71 (2018), pp. 423–455.

[14] Tammy Bahmanziari, J. Michael Pearson, and Leon Crosby. "Is trust important in technology adoption? A policy capturing approach". In: *Journal of Computer Information Systems* 43 (2003), pp. 46–54.

[15] Brian P. Bailey, Joseph A. Konstan, and John V. Carlis. "The effects of interruptions on task performance, annoyance, and anxiety in the user interface". In: *International Conference on Human-Computer Interaction (HCI)*. 2001, pp. 593–601.

[16] Mark Batey and Adrian Furnham. "Creativity, intelligence, and personality: A critical review of the scattered literature." In: *Genetic, Social, and General Psychology Monographs* 132 (2006), pp. 355–429.

[17] Veronika Bauer and Lars Heinemann. "Understanding API usage to support informed decision making in software maintenance". In: *2012 16th European Conference on Software Maintenance and Reengineering*. 2012, pp. 435–440.

[18] Nancy Baym, Limor Shifman, Christopher Persaud, and Kelly Wagman. "Intelligent failures: Clippy memes and the limits of digital assistants". In: *AoIR Selected Papers of Internet Research* (2019).

[19] Fabian Beck, Oliver Moseler, Stephan Diehl, and Günter Daniel Rey. "In situ understanding of performance bottlenecks through visually augmented code". In: *21st International Conference on Program Comprehension (ICPC)*. 2013, pp. 63–72.

[20] Claire Bélisle. "Literacy and the digital knowledge revolution". In: *Digital Literacies for Learning*. 2006, pp. 51–67.

[21] Denys Bernard. "Cognitive interaction: Towards "cognitivity" requirements for the design of virtual assistants". In: *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2017, pp. 210–215.

[22] Barry Boehm. "A view of 20th and 21st century software engineering". In: *28th International Conference on Software Engineering (ICSE)*. 2006, pp. 12–29.

[23] Sebastian K. Boell and Dubravka Cecez-Kecmanovic. "What is an Information System?" In: *48th Hawaii International Conference on System Sciences (HICSS)*. 2015, pp. 4959–4968.

[24] Nathalie Bonnardel and Evelyne Marmèche. "Towards supporting evocation processes in creative design: A cognitive approach". In: *International Journal of Human-Computer Studies* 63 (2005), pp. 422–435.

[25] Nathalie Bonnardel and Franck Zenasni. "The impact of technology on creativity in design: An enhancement?" In: *Creativity and Innovation Management* 19 (2010).

[26] Olimar Borge., Julia Couto, Duncan Ruiz, and Rafael Prikladnicki. "How machine learning has been applied in software engineering?" In: *22nd International Conference on Enterprise Information Systems (ICEIS)*. 2020, pp. 306–313.

[27] Pierre Bourque, Richard E. Fairley, and IEEE Computer Society. *Guide to the software engineering body of knowledge (SWEBOK(R)): Version 3.0*. 2014.

[28] Elodie Bouzekri, Célia Martinie, Philippe Palanque, Katrina Atwood, and Christine Gris. "Should I add recommendations to my warning system? The RCRAFT framework can answer this and other questions about supporting the assessment of automation designs". In: *Human-Computer Interaction (INTERACT)*. 2021, pp. 405–429.

[29] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. "Example-centric programming: integrating web search into the development environment". In: *SIGCHI Conference on Human Factors in Computing Systems*. 2010, pp. 513–522.

[30] Urie Bronfenbrenner. *The ecology of human development: experiments by nature and design*. 1979.

[31] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. "Proactive detection of collaboration conflicts". In: *19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE)*. 2011, pp. 168–178.

[32] Léa Brunschwig, Esther Guerra, and Juan de Lara. "Modelling on mobile devices". In: *Software and Systems Modeling* (2021).

[33] Evgeniy Bryndin. "Collaboration of intelligent interoperable agents Via smart interface". In: *International Journal on Data Science and Technology* 5 (2019), p. 66.

[34] David Budgen. *Software design*. 2003.

[35] David Budgen, Andy J. Burn, Pearl Brereton, Barbara Kitchenham, and Rialette Pretorius. "Empirical evidence about the UML: a systematic literature review". In: *Software: Practice and Experience* 41 (2011), pp. 363–392.

[36] Robin Burke. "Knowledge-based recommender systems". In: *Encyclopedia of library and information systems*. 2000, p. 2000.

[37] Jordi Cabot, Robert Clarisó, Marco Brambilla, and Sébastien Gérard. "Cognifying model-driven software engineering". In: *Software Technologies: Applications and Foundations*. 2018, pp. 154–160.

[38] Liang Cai, Haoye Wang, Qiao Huang, Xin Xia, Zhenchang Xing, and David Lo. "BIKER: a tool for Bi-information source based API method recommendation". In: *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 2019, pp. 1075–1079.

[39] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. "A preliminary analysis on the effects of propensity to trust in distributed software development". In: *IEEE 12th International Conference on Global Software Engineering (ICGSE)*. 2017, pp. 56–60.

[40] Michel R. V. Chaudron, Werner Heijstek, and Ariadi Nugroho. "How effective is UML modeling ?" In: *Software and Systems Modeling* 11 (2012), pp. 571–580.

[41] Kari Chopra and William Alan Wallace. "Trust in electronic environments". In: *36th Annual Hawaii International Conference on System Sciences*. 2003, 10 pp.–-.

[42] Jürgen Cito, Philipp Leitner, Christian Bosshard, Markus Knecht, Genc Mazlami, and Harald C. Gall. "PerformanceHat: augmenting source code with runtime performance traces in the IDE". In: *40th International Conference on Software Engineering: Companion Proceeedings (ICSE)*. 2018, pp. 41–44.

[43] Joel Cordeiro, Bruno Antunes, and Paulo Gomes. "Context-based recommendation to support problem solving in software development". In: *3rd International Workshop on Recommendation Systems for Software Engineering (RSSE)*. 2012, pp. 85–89.

[44] James M. Corrigan. "Augmented intelligence — The new AI — Unleashing human capabilities in knowledge work". In: *34th International Conference on Software Engineering (ICSE)*. 2012, pp. 1285–1288.

[45] Françoise Darses, Françoise Détienne, and Willemien Visser. "Assister la conception: perspectives pour la psychologie cognitive ergonomique". In: *ÉPIQUE 2001, Actes des journées d'étude en psychologie ergonomique* (2019).

[46] Boele De Raad. *The big five personality factors: The psycholexical approach to personality.* 2000.

[47] Brian Dobing and Jeffrey Parsons. "How UML is used". In: *Communications of the ACM* 49 (2006), pp. 109–113.

[48] Kees Dorst and Nigel Cross. "Creativity in the design process: co-evolution of problem–solution". In: *Design Studies* 22 (2001), pp. 425–437.

[49] Ekwa Duala-Ekoko and Martin P. Robillard. "Using structure-based recommendations to facilitate discoverability in APIs". In: *25th European Conference on Object-Oriented Programming (ECOOP)*. 2011, pp. 79–104.

[50] Mickaël Duruisseau, Jean-Claude Tarby, Xavier Le Pallec, and Sébastien Gérard. "VisUML: A live UML visualization to help developers in their programming task". In: *Human Interface and the Management of Information. Interaction, Visualization, and Analytics*. 2018, pp. 3–22.

[51] Ferliana Dwitama and Andre Rusli. "User stories collection via interactive chatbot to support requirements gathering". In: *Telecommunication Computing Electronics and Control (TELKOMNIKA)* 18 (2020), p. 870.

[52] Akil Elkamel, Mariem Gzara, and Hanêne Ben-Abdallah. "An UML class recommender system for software design". In: *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*. 2016, pp. 1–8.

[53] Thomas Erickson, Catalina M. Danis, Wendy A. Kellogg, and Mary E. Helander. "Assistance: The work practices of human administrative assistants and their implications for IT and organizations". In: *ACM Conference on Computer Supported Cooperative Work (CSCW)*. 2008, pp. 609–618.

[54] Linda Erlenhov, Francisco Gomes de Oliveira Neto, Riccardo Scandariato, and Philipp Leitner. "Current and future bots in software development". In: *IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 2019, pp. 7–11.

[55] Robert Feldt, Francisco Gomes de Oliveira Neto, and Richard Torkar. "Ways of applying artificial intelligence in software engineering". In: *6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. 2018, pp. 35–41.

[56] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. "The rise of social bots". In: *Communications of the ACM* 59 (2016), pp. 96–104.

[57] Stephen Fiore, J. Elias, Eduardo Salas, N.W. Warner, and M.P. Letsky. "From data, to information, to knowledge: Measuring knowledge building in the context of collaborative cognition". In: *Macrocognition Metrics and Scenarios: Design and Evaluation for Real-World Teams* (2010), pp. 179–200.

[58] Andrew Forward and Timothy C. Lethbridge. "Problems and opportunities for model-centric versus code-centric software development: A survey of software professionals". In: *International Workshop on Models in Software Engineering (MiSE)*. 2008, pp. 27–32.

[59] Jaroslav Fowkes, Pankajan Chanthirasegaran, Razvan Ranca, Miltiadis Allamanis, Mirella Lapata, and Charles Sutton. "TASSAL: autofolding for source code summarization". In: *38th International Conference on Software Engineering Companion (ICSE)*. 2016, pp. 649–652.

[60] D. Fox and J. L. Kessler. "Experiments in software modeling". In: *Proceedings of the Fall Joint Computer Conference (AFIPS Fall)*. 1967, pp. 429–436.

[61] Marko Gasparic and Andrea Janes. "What recommendation systems for software engineering recommend: A systematic literature review". In: *Journal of Systems and Software* 113 (2016), pp. 101–113.

[62] Andreas Girgensohn, David Redmiles, and Frank M. Shipman. "Agent-based support for communication between developers and users in software design". In: *Ninth Knowledge-Based Software Engineering Conference (KBSE)*. 1994, pp. 22–29.

[63] Vinod Goel and Peter Pirolli. "The structure of design problem spaces". In: *Cognitive Science* 16 (1992), pp. 395–429.

[64] James G. Greeno. "Natures of problem-solving abilities". In: *Handbook of learning and cognitive processes: V. Human information*. 1978, pp. 239–270.

[65] Martin Grossman, Jay E. Aronson, and Richard V. McCarthy. "Does UML make the grade? Insights from the software development community". In: *Inf. Softw. Technol.* 47 (2005), pp. 383–397.

[66] Nielsen Norman Group. *Design Thinking 101*. `https://www.nngroup.com/articles/design-thinking/`.

[67] Jonathan Grudin. "Computer-supported cooperative work: History and focus". In: *Computer* 27 (1994), pp. 19–26.

[68] Jonathan Grudin and Richard Jacques. "Chatbots, humbots, and the quest for artificial general intelligence". In: *Conference on Human Factors in Computing Systems (CHI)*. 2019, pp. 1–11.

[69] Robert J. Hall. "Trusting your assistant". In: *11th Knowledge-Based Software Engineering Conference (KBSE)*. 1996, pp. 42–51.

[70] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. "What would other programmers do: suggesting solutions to error messages". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2010, pp. 1019–1028.

[71] Lile Hattori and Michele Lanza. "Syde: a tool for collaborative software development". In: *32nd ACM/IEEE International Conference on Software Engineering (ICSE)*. 2010, pp. 235–238.

[72] Regina Hebig, Truong Ho Quang, Michel R. V. Chaudron, Gregorio Robles, and Miguel Angel Fernandez. "The quest for open source projects that use UML: Mining GitHub". In: *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2016, pp. 173–183.

[73] Morten Hertzum. "The importance of trust in software engineers' assessment and choice of information sources". In: *Information and Organization* 12 (2002), pp. 1–18.

[74] Thomas T. Hewett. "Informing the design of computer-based environments to support creativity". In: *International Journal of Human-Computer Studies* 63 (2005), pp. 383–409.

[75] Qingning Huo, Hong Zhu, and S. Greenwood. "A multi-agent software engineering environment for testing Web-based applications". In: *27th Annual International Computer Software and Applications Conference (COMPAC)*. 2003, pp. 210–215.

[76] John Hutchinson, Jon Whittle, and Mark Rouncefield. "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure". In: *Science of Computer Programming* 89 (2014), pp. 144–161.

[77] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. "Empirical assessment of MDE in industry". In: *33rd International Conference on Software Engineering (ICSE)*. 2011, pp. 471–480.

[78] Michel Huteau. *Les Conceptions cognitives de la personnalité*. 1985.

[79] ISO. "ISO 9241-210 Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems." In: (2019).

[80] ISO/IEC/IEEE. "Systems and software engineering – Architecture description". In: *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)* (2011), pp. 1–46.

[81] Aníbal Iung, João Carbonell, Luciano Marchezan, Elder Rodrigues, Maicon Bernardino, Fabio Paulo Basso, and Bruno Medeiros. "Systematic mapping study on domain-specific language development tools". In: *Empirical Software Engineering* 25 (2020), pp. 4205–4249.

[82] Yosef Jabareen. "Building a conceptual framework: Philosophy, definitions, and procedure". In: *International Journal of Qualitative Methods* 8 (2009), pp. 49–62.

[83] Nick Jennings and Michael Wooldridge. "Software agents". In: *IEEE Review* 42 (1996), pp. 17–20.

[84] Rodi Jolak. "Understanding and Supporting Software Design in Model-Based Software Engineering". PhD thesis. Chalmers University of Technology and University of Gothenburg, 2020.

[85] John Chris Jones. *Design methods: Seeds of human futures*. 1970.

[86] Thorsten Karrer, Jan-Peter Krämer, Jonathan Diehl, Björn Hartmann, and Jan Borchers. "Stacksplorer: call graph navigation helps increasing code maintenance efficiency". In: *24th annual ACM symposium on User Interface Software and Technology (UIST)*. 2011, pp. 217–224.

[87] Andrew J. Ko and Brad A. Myers. "Extracting and answering why and why not questions about Java program output". In: *ACM Transactions on Software Engineering and Methodology* 20 (2010), 4:1–4:36.

[88] Dimitrios S. Kolovos, Louis M. Rose, Nicholas Matragkas, Richard F. Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, Massimo Tisi, and Jordi Cabot. "A research roadmap towards achieving scalability in Model Driven Engineering". In: *Workshop on Scalability in Model Driven Engineering (BigMDE)*. 2013.

[89] Oleksii Kononenko, David Dietrich, Rahul Sharma, and Reid Holmes. "Automatically locating relevant programming help online". In: *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2012, pp. 127–134.

[90] Terry K. Koo and Mae Y. Li. "A guideline of selecting and reporting intraclass correlation coefficients for reliability research". In: *Journal of Chiropractic Medicine* 15 (2016), pp. 155–163.

[91] Agnes Koschmider, Thomas Hornung, and Andreas Oberweis. "Recommendation-based editor for business process modeling". In: *Data & Knowledge Engineering* 70 (2011), pp. 483–503.

[92] Carine Lallemand and Guillaume Gronier. *Méthodes de design UX: 30 méthodes fondamentales pour concevoir des expériences optimales*. 2nd ed. 2018.

[93] Craig Larman. *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. 2012.

[94] Thomas D. LaToza and Brad A. Myers. "Visualizing call graphs". In: *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2011, pp. 117–124.

[95] Brian Lee, Savil Srivastava, Ranjitha Kumar, Ronen Brafman, and Scott R. Klemmer. "Designing with interactive example galleries". In: *SIGCHI Conference on Human Factors in Computing Systems*. 2010, pp. 2257–2266.

[96] Seonah Lee, Sungwon Kang, and Matt Staats. "NavClus: A graphical recommender for assisting code exploration". In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 1315–1318.

[97] Timothy C. Lethbridge. "What knowledge is important to a software professional?" In: *Computer* 33 (2000), pp. 44–50.

[98] Tom Lieber, Joel R. Brandt, and Rob C. Miller. "Addressing misconceptions about code with always-on programming visualizations". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2014, pp. 2481–2490.

[99] Tilmann Lindberg, Christoph Meinel, and Ralf Wagner. "Design thinking: A fruitful concept for IT development?" In: *Design Thinking: Understand – Improve – Apply*. 2011, pp. 3–18.

[100] Robert B. Lount Jr. and Nathan C. Pettit. "The social context of trust: The role of status." In: *Organizational Behavior and Human Decision Processes* 117 (2012), pp. 15–23.

[101] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. "Recommender system application developments: A survey". In: *Decision Support Systems* 74 (2015), pp. 12–32.

[102] Todd Lubart, Christophe Mouchiroud, Sylvie Tordjman, and Franck Zenasni. *Psychologie de la créativité*. 2003.

[103] Daniel Lucrédio, Renata P. de M. Fortes, and Jon Whittle. "MOOGLE: a metamodel-based model search engine". In: *Software and Systems Modeling* 11 (2012), pp. 183–208.

[104] Alexander Maedche, Christine Legner, Alexander Benlian, Benedikt Berger, Henner Gimpel, Thomas Hess, Oliver Hinz, Stefan Morana, and Matthias Söllner. "AI-based digital assistants". In: *Business and Information Systems Engineering* 61 (2019), pp. 535–544.

[105] Alexander Maedche, Stefan Morana, Silvia Schacht, Dirk Werth, and Julian Krumeich. "Advanced user assistance systems". In: *Business and Information Systems Engineering* 58 (2016), pp. 367–370.

[106] Thomas W. Malone. "How can human-computer "Superminds" develop business strategies?" In: *The Future of Management in an AI World: Redefining Purpose and Strategy in the Fourth Industrial Revolution*. 2020, pp. 165–183.

[107] Thomas W. Malone and Michael S. Bernstein. *Handbook of Collective Intelligence*. 2015.

[108] Nikos Manouselis and Constantina Costopoulou. "Analysis and classification of multi-criteria recommender systems". In: *World Wide Web* 10 (2007), pp. 415–441.

[109] Dominique Maurel and Aïda Chebbi. "La perception de la confiance informationnelle. Impacts sur les comportements informationnels et les pratiques documentaires en contexte organisationnel". In: *Communication and Organisation* 42 (2012), pp. 73–90.

[110] Collin McMillan, Negar Hariri, Denys Poshyvanyk, Jane Cleland-Huang, and Bamshad Mobasher. "Recommending source code for use in rapid software prototypes". In: *34th International Conference on Software Engineering (ICSE)*. 2012, pp. 848–858.

[111] Michael F. McTear. "The rise of the conversational interface: A new kid on the block?" In: *Future and Emerging Trends in Language Technology. Machine Learning and Big Data*. 2017, pp. 38–49.

[112] Kim Mens and Angela Lozano. "Source code-based recommendation systems". In: *Recommendation Systems in Software Engineering*. 2014, pp. 93–130.

[113] Pedro Meseguer. "Towards a conceptual framework for expert system validation". In: *AI Communication* 5 (1992), pp. 119–135.

[114] Alan S. Miller and Tomoko Mitamura. "Are Surveys on Trust Trustworthy?" In: *Social Psychology Quarterly* 66 (2003), pp. 62–70.

[115] Mohamed Wiem Mkaouer, Marouane Kessentini, Slim Bechikh, Kalyanmoy Deb, and Mel Ó Cinnéide. "Recommendation system for software refactoring using innovization and interactive dynamic optimization". In: *29th ACM/IEEE international conference on Automated software engineering (ASE)*. 2014, pp. 331–336.

[116] Gail C. Murphy. "Beyond integrated development environments: Adding context to software development". In: *IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. 2019, pp. 73–76.

[117] Emerson Murphy-Hill and Gail C. Murphy. "Recommendation delivery - Getting the user interface just right". In: *Recommendation Systems in Software Engineering*. 2014.

[118] Kivanç Muşlu, Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. "Improving IDE recommendations by considering global implications of existing recommendations". In: *34th International Conference on Software Engineering (ICSE)*. 2012, pp. 1349–1352.

[119] Gunter Mussbacher, Benoit Combemale, Jörg Kienzle, Silvia Abrahão, Hyacinth Ali, Nelly Bencomo, Márton Búr, Loli Burgueño, Gregor Engels, Pierre Jeanjean, Jean-Marc Jézéquel, Thomas Kühn, Sébastien Mosser, Houari Sahraoui, Eugene Syriani, Dániel Varró, and Martin Weyssow. "Opportunities in intelligent modeling assistance". In: *Software and Systems Modeling* (2020).

[120] Farzaneh Nasirian, Mohsen Ahmadian, and One-Ki (Daniel) Lee. "AI-based voice assistant systems: Evaluating from the interaction and trust perspectives". In: *23rd Americas Conference on Information Systems (AMCIS)*. 2017.

[121] Anh Tuan Nguyen, Hoan Anh Nguyen, Tung Thanh Nguyen, and Tien N. Nguyen. "GraPacc: A graph-based pattern-oriented, context-sensitive code completion tool". In: *34th International Conference on Software Engineering (ICSE)*. 2012, pp. 1407–1410.

[122] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. "CrossRec: Supporting software developers by recommending third-party libraries". In: *Journal of Systems and Software* 161 (2020), p. 110460.

[123] Jakob Nielsen. "End of web design". In: *Nielsen Norman Group* (2000).

[124] John O'Donovan and Barry Smyth. "Trust in recommender systems". In: *10th international conference on Intelligent User Interfaces (IUI)*. 2005, pp. 167–174.

[125] Manuel Ohrndorf, Christopher Pietsch, Udo Kelter, and Timo Kehrer. "ReVision: a tool for history-based model repair recommendations". In: *40th International Conference on Software Engineering: Companion Proceeedings (ICSE)*. 2018, pp. 105–108.

[126] Harold Ossher, William Harrison, and Peri Tarr. "Software engineering tools and environments: A roadmap". In: *Conference on The Future of Software Engineering*. 2000, pp. 261–277.

[127] *Oxford English dictionary (Online)*.

[128] Mert Ozkaya and Ferhat Erata. "Understanding practitioners' challenges on software modeling: A survey". In: *Journal of Computer Languages* 58 (2020).

[129] Philippe Palanque. "Ten objectives and ten rules for designing automations in interaction techniques, user interfaces and interactive systems". In: *Proceedings of the International Conference on Advanced Visual Interfaces*. 2020.

[130] Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. "A model for types and levels of human interaction with automation". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 30 (2000), pp. 286–297.

[131] Chris Parnin and Spencer Rugaber. "Resumption strategies for interrupted programming tasks". In: *IEEE 17th International Conference on Program Comprehension*. 2009, pp. 80–89.

[132] Michael J. Pazzani and Daniel Billsus. "Content-based recommendation systems". In: *The Adaptive Web: Methods and Strategies of Web Personalization*. 2007, pp. 325–341.

[133] Kari Pearlson and Carol Saunders. *Managing  using information systems: A strategic approach*. 2013.

[134] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. "Collaborative modeling and group decision making using chatbots in social networks". In: *IEEE Software* 35 (2018), pp. 48–54.

[135] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. "Guidelines for conducting systematic mapping studies in software engineering: An update". In: *Information and Software Technology* 64 (2015), pp. 1–18.

[136] Marian Petre. "UML in practice". In: *International Conference on Software Engineering (ICSE)*. 2013, pp. 722–731.

[137] Elena Planas and Jordi Cabot. "How are UML class diagrams built in practice? A usability study of two UML tools: Magicdraw and Papyrus". In: *Computer Standards  Interfaces* 67 (2020).

[138] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. "Mining StackOverflow to turn the IDE into a self-confident programming prompter". In: *11th Working Conference on Mining Software Repositories (MSR)*. 2014, pp. 102–111.

[139] Luca Ponzanelli, Simone Scalabrino, Gabriele Bavota, Andrea Mocci, Rocco Oliveto, Massimiliano Di Penta, and Michele Lanza. "Supporting software developers with a holistic recommender system". In: *IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, pp. 94–105.

[140] Javier Portillo-Rodríguez, Aurora Vizcaíno, Mario Piattini, and Sarah Beecham. "Tools used in global software engineering: A systematic mapping review". In: *Information and Software Technology* 54 (2012), pp. 663–685.

[141] Pearl Pu and Li Chen. "Trust building with explanation interfaces". In: *11th International Conference on Intelligent User Interfaces (IUI)*. 2006, pp. 93–100.

[142] Pearl Pu, Li Chen, and Rong Hu. "A user-centric evaluation framework for recommender systems". In: *5th ACM conference on Recommender systems (RecSys)*. 2011, pp. 157–164.

[143] Mohammad Masudur Rahman, Shamima Yeasmin, and Chanchal K. Roy. "Towards a context-aware IDE-based meta search engine for recommendation about programming errors and exceptions". In: *IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. 2014, pp. 194–203.

[144] Paul Ralph. "Comparing two software design process theories". In: *Global Perspectives on Design Science Research*. 2010, pp. 139–153.

[145] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. *Design principles for tools to support creative thinking*. Tech. rep. 2005.

[146] Martin P. Robillard and Robert J. Walker. "An introduction to recommendation systems in software engineering". In: vol. Recommendation Systems in Software Engineering. 2014, pp. 1–11.

[147] Jennifer Rowley. "The wisdom hierarchy: representations of the DIKW hierarchy". In: *Journal of Information Science* 33 (2007), pp. 163–180.

[148] Bernard Roy. *Multicriteria methodology for decision aiding*. 2013.

[149] Yong Rui. "From artificial intelligence to augmented intelligence". In: *IEEE MultiMedia* 24 (2017), pp. 4–5.

[150] Mark A. Runco and Steven R. Pritzker. *Encyclopedia of creativity*. 1999.

[151] Ioana Rus. "Guest editors' introduction: Knowledge management in software engineering". In: *IEEE Software* 19 (2002), pp. 26–38.

[152] Safdar Aqeel Safdar, Muhammad Zohaib Iqbal, and Muhammad Uzair Khan. "Empirical evaluation of UML modeling tools–A controlled experiment". In: *Modelling Foundations and Applications*. 2015, pp. 33–44.

[153] Ripon K. Saha, Hiroaki Yoshida, Mukul R. Prasad, Susumu Tokumoto, Kuniharu Takayama, and Isao Nanba. "Elixir: an automated repair tool for Java programs". In: *40th International Conference on Software Engineering: Companion Proceeedings (ICSE)*. 2018, pp. 77–80.

[154] Rosa San Segundo. "A new concept of knowledge". In: *Online Information Review* 26 (2002), pp. 239–245.

[155] André L. Santos, Gonçalo Prendi, Hugo Sousa, and Ricardo Ribeiro. "Stepwise API usage assistance using n-gram language models". In: *Journal of Systems and Software* 131 (2017), pp. 461–474.

[156] Maxime Savary-Leblanc, Loli Burgueño, Xavier Le-Pallec, Sébastien Gérard, and Jordi Cabot. *SLR data tables and resources. https://software-assistant.univ-lille.fr*. 2021.

[157] Maxime Savary-Leblanc, Xavier Pallec, and Sébastien Gérard. "A recommender system to assist conceptual modeling with UML". In: *International Conference on Software Engineering and Knowledge Engineering (SEKE)*. 2021.

[158] Dmitrii Savchenko, Jussi Kasurinen, and Ossi Taipale. "Smart tools in software engineering: A systematic mapping study". In: *42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2019, pp. 1509–1513.

[159] Nicholas Sawadsky, Gail C. Murphy, and Rahul Jiresal. "Reverb: Recommending code-related web pages". In: *35th International Conference on Software Engineering (ICSE)*. 2013, pp. 812–821.

[160] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. "Collaborative filtering recommender systems". In: *The Adaptive Web: Methods and Strategies of Web Personalization*. 2007, pp. 291–324.

[161] Gabriel Sebastián, Jose A. Gallud, and Ricardo Tesoriero. "Code generation using model driven architecture: A systematic mapping study". In: *Journal of Computer Languages* 56 (2020).

[162] Ángel Mora Segura and Juan de Lara. "Extremo: An Eclipse plugin for modelling and meta-modelling assistance". In: *Science of Computer Programming* 180 (2019), pp. 71–80.

[163] Nigel Shadbolt, Enrico Motta, and A. Rouge. "Constructing knowledge-based systems". In: *IEEE Software* 10 (1993), pp. 34–38.

[164] Saad Shafiq, Atif Mashkoor, Christoph Mayr-Dorn, and Alexander Egyed. *Machine learning for software engineering: A systematic mapping*. 2020.

[165] Patrick C. Shih, Gina Venolia, and Gary M. Olson. "Brainstorming under constraints: why software developers brainstorm in groups". In: *25th BCS Conference on Human-Computer Interaction (BCS-HCI)*. 2011, pp. 74–83.

[166] *Proceedings of the 1st International Workshop on Bots in Software Engineering (BotSE)*. 2019.

[167] Ben Shneiderman. "Creating Creativity: User Interfaces for Supporting Innovation". In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 7 (2000), pp. 114–138.

[168] Alejandra Siles Antezana. "TOAD: A tool for recommending auto-refactoring alternatives". In: *IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE)*. 2019, pp. 174–176.

[169] Eric Simon. "La confiance dans tous ses états". In: *Revue française de gestion* 33 (2007).

[170] Herbert A. Simon. "The structure of ill structured problems". In: *Artificial Intelligence* 4 (1973), pp. 181–201.

[171] *Handbook of creativity.* 1999.

[172] Robert J. Sternberg, Todd I. Lubart, James C. Kaufman, and Jean E. Pretz. "Creativity." In: *The Cambridge handbook of thinking and reasoning.* 2005, pp. 351–369.

[173] Margaret-Anne Storey, Alexander Serebrenik, Carolyn Penstein Rosé, Thomas Zimmermann, and James D. Herbsleb. "BOTse: Bots in software engineering (Dagstuhl Seminar 19471)". In: *Dagstuhl Reports* 9 (2020), pp. 84–96.

[174] Margaret-Anne Storey and Alexey Zagalsky. "Disrupting developer productivity one bot at a time". In: *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. 2016, pp. 928–931.

[175] Didi Surian, Nian Liu, David Lo, Hanghang Tong, Ee-Peng Lim, and Christos Faloutsos. "Recommending people in developers' collaboration network". In: *2011 18th Working Conference on Reverse Engineering (WCRE)*. 2011, pp. 379–388.

[176] Ben Swift, Andrew Sorensen, Henry Gardner, and John Hosking. "Visual code annotations for cyberphysical programming". In: *1st International Workshop on Live Programming (LIVE)*. 2013, pp. 27–30.

[177] Watanabe Takuya and Hidehiko Masuhara. "A spontaneous code recommendation tool based on associative search". In: *3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation (SUITE)*. 2011, pp. 17–20.

[178] Antony Tang and Hans van Vliet. "Modeling constraints improves software architecture design reasoning". In: *Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture (WICSA-ECSA)*. 2009, pp. 253–256.

[179] Cédric Teyton, Jean-Rémy Falleri, Floréal Morandat, and Xavier Blanc. "Find your library experts". In: *20th Working Conference on Reverse Engineering (WCRE)*. 2013, pp. 202–211.

[180] Andreas Thies and Christian Roth. "Recommending rename refactorings". In: *2nd International Workshop on Recommendation Systems for Software Engineering (RSSE)*. 2010, pp. 1–5.

[181] John C. Thomas, Alison Lee, and Catalina Danis. "Enhancing creative design via software tools". In: *Communications of the ACM* 45 (2002), pp. 112–115.

[182] Nava Tintarev and Judith Masthoff. "A survey of explanations in recommender systems". In: *2007 IEEE 23rd International Conference on Data Engineering Workshop*. 2007, pp. 801–810.

[183] Suzanne Tolmeijer, Ujwal Gadiraju, Ramya Ghantasala, Akshit Gupta, and Abraham Bernstein. "Second chance for a first impression? Trust development in intelligent system interaction". In: *29th ACM Conference on User Modeling, Adaptation and Personalization (UMAP)*. 2021, pp. 77–87.

[184] Sven Tuzovic. "Talk to me – The rise of voice assistants and smart speakers: A balance between efficiency and privacy". In: *Handbook of Digital Marketing and Social Media*. 2021.

[185] Marialena Vagia, Aksel A. Transeth, and Sigurd A. Fjerdingen. "A literature review on the levels of automation during the years. What are the different taxonomies that have been proposed?" In: *Applied Ergonomics* 53 (2016), pp. 190–202.

[186] Ragnhild Van Der Straeten, Tom Mens, and Stefan Van Baelen. "Challenges in model-driven software engineering". In: *Models in Software Engineering*. 2009, pp. 35–47.

[187] Petcharat Viriyakattiyaporn and Gail C. Murphy. "Improving program navigation with an active help system". In: *Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*. 2010, pp. 27–41.

[188] Kai Wang and Jeffrey V. Nickerson. "A literature review on individual creativity support systems". In: *Computers in Human Behavior* 74 (2017), pp. 139–151.

[189] Lijie Wang, Lu Fang, Leye Wang, Ge Li, Bing Xie, and Fuqing Yang. "APIExample: An effective web search based usage example recommendation system for java APIs". In: *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2011, pp. 592–595.

[190] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. "Industrial adoption of Model-Driven Engineering: Are the tools really the problem?" In: *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2013, pp. 1–17.

[191] Michael Wooldridge. "Intelligent agents: The key concepts". In: *Multi-Agent Systems and Applications II*. 2002, pp. 3–43.

[192] Congying Xu, Bosen Min, Xiaobing Sun, Jiajun Hu, Bin Li, and Yucong Duan. "MULAPI: A tool for API method and usage location recommendation". In: *IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE)*. 2019, pp. 119–122.

[193] Yasuhiro Yamamoto and Kumiyo Nakakoji. "Interaction design of tools for fostering creativity in the early stages of information design". In: *International Journal of Human-Computer Studies* 63 (2005), pp. 513–535.

[194] Zhiqiang Yan, Remco Dijkman, and Paul Grefen. "Business process model repositories – Framework and survey". In: *Information and Software Technology* 54 (2012), pp. 380–395.

[195] Weizhao Yuan, Hoang H. Nguyen, Lingxiao Jiang, and Yuting Chen. "LibraryGuru: API recommendation for Android developers". In: *40th International Conference on Software Engineering: Companion Proceeedings (ICSE)*. 2018, pp. 364–365.

[196] Alexey Zagalsky, Ohad Barzilay, and Amiram Yehudai. "Example Overflow: Using social media for code recommendation". In: *3rd International Workshop on Recommendation Systems for Software Engineering (RSSE)*. 2012, pp. 38–42.

[197] Cheng Zhang, Juyuan Yang, Yi Zhang, Jing Fan, Xin Zhang, Jianjun Zhao, and Peizhao Ou. "Automatic parameter recommendation for practical API usage". In: *34th International Conference on Software Engineering (ICSE)*. 2012, pp. 826–836.

# Index

Information about the concepts listed below can be found at the indicated pages.

# Contents