An analysis of software design understanding & motivation of engineering students

Jean-Christophe Bach, Antoine Beugnard, Jean-Loup Castaigne, Julien Mallet, Salvador Martínez, Maria-Teresa Segarra

An analysis of software design understanding & motivation of engineering students

Jean-Christophe Bach
IMT Atlantique
Lab-STICC, UMR 6285
Brest, France
jc.bach@imt-atlantique.fr

Antoine Beugnard
IMT Atlantique
Lab-STICC, UMR 6285
Brest, France
antoine.beugnard@imt-atlantique.fr

Jean-Loup Castaigne
EVS-LAURe, UMR 5600
IMT Atlantique
Nantes, France
jean-loup.castaigne@imt-atlantique.fr

Julien Mallet
IMT Atlantique
Lab-STICC, UMR 6285
Brest, France
julien.mallet@imt-atlantique.fr

Salvador Martínez
IMT Atlantique
Lab-STICC, UMR 6285
Brest, France
salvador.martinez@imt-atlantique.fr

Maria-Teresa Segarra
IMT Atlantique
Lab-STICC, UMR 6285
Brest, France
mt.segarra@imt-atlantique.fr

ABSTRACT

Software engineering is now a well-recognized discipline in higher education. One challenge of software engineering education is to design teaching activities that empower students motivation and engagement. In this paper we present a study aimed at evaluating the understanding, motivation and perception of students with respect to software design activities. We first describe a project-based course currently taught to second-year students at IMT Atlantique, a French engineering school, and then present a questionnaire-based experiment. We asked the 44 students enrolled to the aforementioned course in the 2021 edition to answer three questionnaires (before and after the design activities and at the end of the course). The objective of this experiment is threefold: 1) evaluate the student's perception of the usefulness of the software design phase and the activity of modeling of software systems; 2) assess the evolution, if any, of this perception along the course; 3) evaluate if our course fosters student engagement. Results show that students are motivated with a good engagement all along the project and the perception of usefulness and pertinence of software design evolves in a slightly positive way.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; • **Software and its engineering** → *Designing software.*

KEYWORDS

software design, student engagement, empirical study

1 INTRODUCTION

Software engineering is now a well-recognized discipline in higher education. Students enrolled in software engineering courses learn how to design, build, test, and maintain software products [2]. One of the challenges of software engineering education is to design teaching activities that empower students motivation and engagement [11]. One effective way to achieve this is to propose project-based learning activities [8]. Indeed, while traditional lessons (e.g., lectures, recitations, and labs) may be used to provide students the required theoretical knowledge, activities where students are actively involved in the design and implementation of a software solution to a given problem are known to be more engaging [1, 6]. Coming up with a project that allows students to actively work on a software solution, fosters engagements and produces good learning outcomes may not be very difficult *per se*. However, academic projects tend to be small, short-term and defined in a controlled environment [10], which not really require software engineering concepts, methods and tools. As a consequence, it is difficult to determine if the students *really* understand the importance of software engineering goals, concepts and practices.

To set the context, students at IMT Atlantique, a French engineering, school follow a common first year devoted to basic engineering foundations. Afterwards, the students specialize in different engineering fields such as telecommunications, energy, electronics, industry or computer science. The aforementioned common first year includes 90 hours of training on the basic engineering foundations of computer science. From these 90 hours, 30 are devoted to object-oriented programming in Java and basic UML notations. More than 1/3 of the 90 hours in the first year are dedicated to project-based learning activities. This ratio is maintained in the second and third year for the students that specialize in computer science. Although different stages of a software product life cycle are addressed in our projects, we pay special attention to the design stage so as to convince students that reasoning on models is key to control software quality.

Our observation, after more than 10 years of applying this approach, is still uncertain. Some students appear to be convinced of the interest of software design in the early steps of the projects, others seem to understand the interest only when they make the

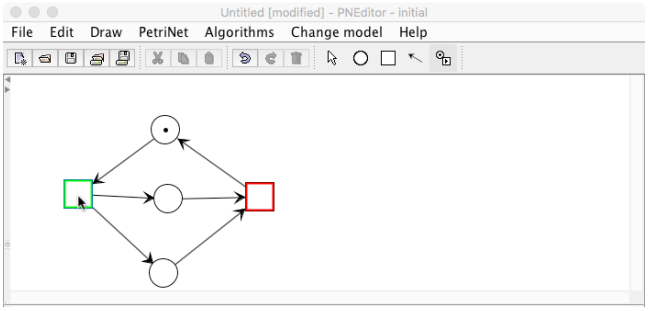

Figure 2: PNEditor graphical interface

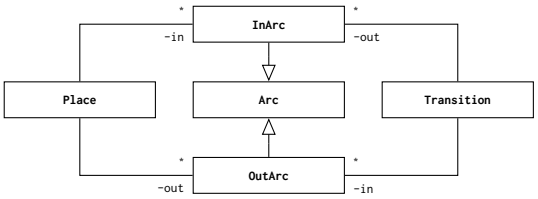

Figure 3: An example of an initial design

connection between the code and the model (after the code is produced), a few appear to remain unconvinced. This uncertainty leads us to undertake a more controlled experiment. In this sense, we have designed a questionnaire-based experiment with students enrolled in a second-year course at IMT Atlantique. The goal of this experiment is three-fold:

- **G1** Evaluate the students perception of the usefulness and pertinence of the software design phase and the activity of modeling of software systems.
- **G2** Assess the evolution, if any, of this perception along the course.
- **G3** Evaluate if our approach fosters student engagement.

In this paper we present this experiment and our analysis of the results.

The rest of the paper is structured as follows. Section 2 presents the structure of the course. Section 3 describes the experiment and an overview of the questionnaires submitted to the students. Section 4 contains the results and the analysis of the experiment. In Section 5, we discuss related work. We conclude and give some insights on future work in Section 6. The questionnaires are in the Appendix A.

2 COURSE DESCRIPTION

The experiment was conducted with students on the second year of IMT Atlantique engineering school. Their common knowledge and skills in computer science are those developed in the first year at IMT Atlantique (last year bachelor's degree), where 90 hours are devoted to the discovery of programming through: graph theory in Python (25 hours); object-oriented programming in Java together with basic UML notations (30 hours); and combined human-machine interface (HMI) and database foundations (35 hours).

In this context, we applied our experiment to a 60-hour course dedicated to introduce object-oriented (implemented in Java) and software engineering concepts. Object-oriented concepts concern 40 hours of traditional lectures which include interfaces, exceptions, types and genericity, lambda expressions and streams. Software engineering concepts are mainly introduced in a 22-hours-long project (μP) organized as a simplified V life-cycle software development project where students work in pairs. Design, implementation and tests, and integration phases are sequentially covered in the μP (see figure 1). Analysis was discarded as we considered that teachers know exactly what they want. Although an agile or iterative life-cycle process may be better suited for this project, we believe a V life-cycle is better adapted to a sequential agenda where each time slot is clearly associated with a phase. Understanding of the roles, objectives and tools of each phase is eased as well.

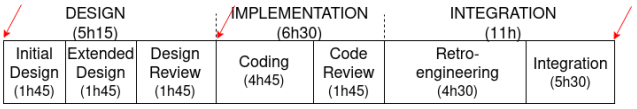

Figure 1: Stages of the μP

As in [15] and [7], the project consists in the implementation of a Petri net model [13]. Additionally, students are required to integrate their model with an existing graphical interface (see figure 2). A Petri net is a simple mathematical model that can be implemented using a few classes. It offers limited but significant variability so that object-oriented overriding and late binding appear as natural mechanisms to deal with it. The graphical interface is an adaptation of the open-source project PNEditor[1], that has been modified to accept different Petri net models by using the *Adapter* design pattern [9]. A specific menu allows users to switch from one model to another. Two default models of a Petri net are included in PNEditor so that students can run them in order to better understand how a Petri net works.

A 16-page document briefly introduces the main characteristics of a basic Petri net model, and the goal of the μP. Then, it describes each phase, including its duration, goals, and outcomes.

The agenda, and development phases are as follows:

Design (5h15). In this phase, students use UML models to represent structural (class diagrams) and temporal behavior (sequence diagrams) of their solution to a given problem (Petri net). Teachers give advice, and ask questions but do not dictate (neither prescribe) a solution even if they can anticipate implementation issues. In order to highlight the consequences of a *poor* model, the phase is divided into three well-defined activities (represented in figure 1):

- Initial Design (1h45). In this activity, students should design a basic Petri net. Both, structural and temporal models have to be delivered and made choices explained in terms of their advantages and drawbacks. An example of a class diagram provided by students as their initial design is shown in figure 3.
- Extended design (1h45). Requirements of the Petri net are modified: inhibitor arcs and reset arcs may be used. Again, the class diagram, a sequence diagram representing how

[1] https://pneditor-org-hrd.appspot.com

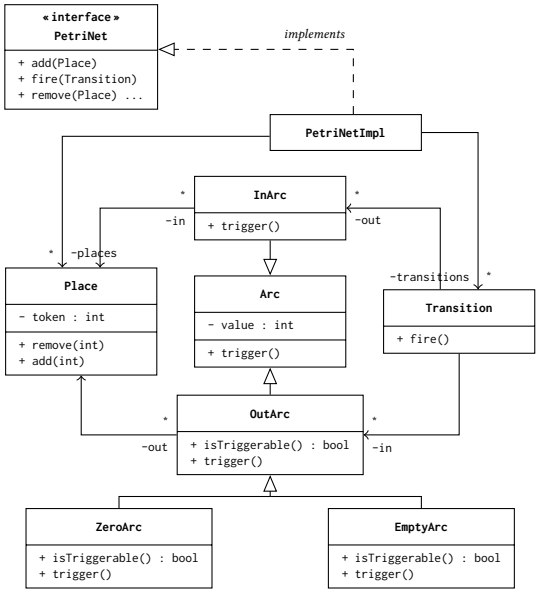

Figure 4: An example of a design for Petri net extensions

the model works when a transition is fired, and explanations for the choices made should be provided by students. Depending on the initial design, taking into account the new requirements may be straightforward or a challenge leading to an entirely new model. Figure 4 shows a class diagram resulting from the extension of Figure 3's model.

- Design Review (1h45). Teachers select two to four models produced by students to be discussed by the whole class in terms of the concepts they reified, the relationships between concepts, and the identified methods (services). Students can then reconsider their model in order to take into account lessons learned from the activity.

Implementation (6h30). In the implementation phase, students proceed to write Java code to implement their model. The resulting code may not be consistent with the model, but students should explain the reasons of the inconsistencies. Once the code is finished and tested, students exchange their code, and review it. The code, along with the inclusion (or absence) of comments and the respect for good programming practices are open to positive evaluation or constructive criticism.

Integration (11h). Once their model is implemented, students are asked to integrate it with the PNEditor. First, they are invited to explore its source code in order to identify how their code may be integrated with the PNEditor. Although they are guided by teachers, much of the work is done by using a particular tool, STAN[2], a structure analyzer for Java. Once the main components of the PNEditor are understood, the adapter design pattern is used.

As seen from figure 1, software design and modeling is an important task all along the project: considering the whole design phase and the reverse engineering stage during the integration phase, it comprises a total of 9h45 from the 22h of the project. During the design phase, the introduction of the new requirements makes students focus on the importance of software evolution when considering a solution. During the integration phase, software models are used to help with the comprehension of a given code (PNEditor). In neither phase, teachers give ready-to-go solutions to students: they act as guides to help students identify their own solutions even if they may lead to implementation issues.

[2]https://marketplace.eclipse.org/content/stan-structure-analysis-java

3 EXPERIMENT DESCRIPTION

Two groups of students (A & D) participated in the experiment. They had slightly different profiles. Group A concerned students who specialized in software engineering, while students in group D specialized in other engineering domains that need some foundations on software engineering such as collaborative HMI, embedded systems or smart objects. Despite these different profiles, the organization of the project-related activities was similar : a simplified V life-cycle software development project that covers design, implementation, tests, and integration of a Petri net model. The main difference concerns the amount of time dedicated to the project: +20% of time for students in group D. The rationale behind this is that, as for the students in group D the main focus is not software development itself, they may need more time to achieve a similar result as students in group A. However, we believe such (small) differences should not influence the results of the experiment. We verify this intuition in Section 4.

The experiment was designed to provide us with insights on whether and how project-related activities impact the understanding of software design. Therefore, it is structured as a three-staged survey (see red arrows in the figure 1). We collected information:

- Before starting any project-related activity in order to identify students that are already familiar with software design. For these students, our course may not bring significant improvement on their knowledge.
- After the design phase, but before implementation, to understand if design activities have a positive impact on the understanding of the importance of software design.
- At the end of the project when the impact of design choices become apparent on the product code.

Each stage consisted in a 5–10 minutes Moodle individual questionnaire. For anonymity purposes, each student was identified by a randomly selected number. This number allowed us to associate responses from one student to the same question at different stages. In particular, a set of five questions on software design was included in all three questionnaires to observe the evolution of software design understanding (goals G1 & G2):

(1) When should software design take place during development life-cycle?
(2) What are the expected results (artifacts) of software design? (in a few words)
(3) What are the benefits of software design?
(4) What are the drawbacks of software design?
(5) List a few tools used during software design? (do not restrict your answers to software tools)

In addition to these five design-related questions (*D*-series questions in the appendices) each stage included a set of four to five

%	B2	B3	B4
++	39	0	71
+	52	93	18
-	9	5	11
- -	0	2	0

Table 1: Answers to background questions (in percentage)

%	M1	M2	M3
strongly agree	83	74	51
agree	9	20	40
disagree	3	0	3
strongly disagree	5	6	6

Table 2: Answers to motivation questions (in percentage)

other questions with different goals depending on the questionnaire:

- Questionnaire 1 collects general information like student background, interest in CS or programming skills self assessment (*B*-series questions in appendices)
- Questionnaire 2 focuses on student engagement (goal G3) on software design activities (*M*-series questions in appendices). For these questions, we used the motivation framework proposed by Viau ([12]) that identifies three axis to consider an activity as engaging: the value of the activity for the student; the confidence of the student to successfully finish it; and, the challenge it poses to the student.
- The last questionnaire completes the *D*-series questions (goals G1 & G2) with a level of confidence. Finally it asks students whether they are convinced that software design must precede programming and if so, when they realized about that (*F*-series questions).

More details on the questionnaires are included in appendices.

4 RESULTS AND ANALYSIS

We had 44 students answering the three questionnaires in two groups (A & D). We could collect only 10 + 7 "full answers", respectively. A "full answer" is a sequence of three questionnaire answers from the same (anonymous) student. We analyze results in two different ways: the 44 answers per questionnaire without the time aspect, and the 17 timed-and-sequenced-answers ("full answers"). The first one gives us information concerning perception of software design usefulness and student engagement, while the second one gives us insights on the evolution of both.

Table 1 shows the profile of the 44 students (in percentage) regarding their interest in computer science (column B2), self-estimated skills in programming (column B3), and intended engagement in the course (column B4). As seen in the table, the student sample is homogeneous and very broadly interested in computer science and the course itself (++ and + responses). Their self-estimated programming skills are also homogeneous; 93% are mostly able to write small object-oriented programs.

4.1 Results

Table 2 presents the results of the students' motivation for software design activities. Column M1 concerns interest in the activity (92 % agree or strongly agree). Column M2 deals with the students' ability to succeed the activity (94 % agree or strongly agree). Column M3 addresses the perceived students' challenge for the activity (91 % agree or strongly agree). This result clearly shows that software design activities proposed in the course satisfy the Viau three axis for an engaging activity [12]: value (interest), confidence, and challenge.

Table 3 presents the results concerning the understanding of software design and its evolution from questionnaire 1 to questionnaire 3. The table includes answers to 5 questions (columns D1 to D5) and, for each question, three sub-columns, one per questionnaire. Four types of answers are possible: ✓ if the answer is correct, **?** if the student does not know, ✗ if the answer is not correct, and - if the student answered but the response is ambiguous.

The answers to the question associated with column D1 ("when should software design take place during a development life-cycle?") are circa 90% correct. In addition, there was little variation over time (from questionnaire 1 to questionnaire 3). Therefore, when is that the software design activity must take place within the development life-cycle seems clear to a vast majority of students.

Regarding the questions associated with columns D2 to D5 (expected outcomes of software design, benefits and drawbacks, and tools to be used during design, respectively), the initial correct answers (questionnaire 1) are below or much below 50 %. They are even lower (18%) w.r.t. the question associated with column D4. However, after software design activities, this number increases by 15 to 30 points. For the question associated with columns D2 and D3 (expected outcomes and benefits of software design, respectively), the number of correct answers continues to grow for the third questionnaire while for D5 (tools for software design) and especially D4 (drawbacks of software design), it is not the case.

This can also be seen in table 4. This table presents results concerning the 17 "full answers" collected in our experiment. Column # identifies the student. Columns D1 to D5 are the answers to the same questions discussed in table 3. A new question from the third questionnaire is presented in the table (F2): "when did you realize that software design must precede programming?". Numbers in this column correspond to types of activities in the course: non-μP activities (number 2), μP activities (software design (3), implementation (4), and integration (5)), when answering the survey (number 6). Number 1 stands for "before the beginning of the course".

Desirable trends (from **?** ("don't know") or ✗ ("incorrect answer") to ✓ ("correct answer")) for each D-series question (columns D1 to D5) are presented in green in the table. We identify such trends between the first and the second questionnaire (for example, student #3 for the question associated with column D2) and also between the first and third one (for example, student #1 for the question associated with column D3). Also, the 17 "full-answered" students know when software design should take place from the first questionnaire (results in column D1). However, this result is not always consistent with answers in column F2 (e.g., student #2 indicates that the final questionnaire helped her to understand it).

Finally, for columns D2 to D5, there are 68 (17×4) "full answers": 23 (34%) are correct (*i.e.*, answers are always ✓ in the table) and 21 (31%) represent a desirable trend (11 (52%) from the first to

%	D1			D2			D3			D4			D5		
✓	84	89	87	32	51	57	43	57	63	18	46	30	48	71	70
?	7	0	0	48	26	13	32	26	10	57	40	13	11	14	7
✗	7	11	13	20	23	20	18	9	23	16	6	23	20	11	17
-	2	0	0	0	0	10	7	8	4	9	8	34	21	4	6

Table 3: Answers to software design related questions (in percentage)

#	D1			D2			D3			D4			D5			F2
1	✓	✓	✓	?	?	✓	?	?	✓	?	?	✓	✗	✗	✓	1
2	✓	✓	✓	?	✗	✓	?	?	?	?	?	?	✓	-	-	6
3	✓	✓	✓	?	✓	✗	✓	✓	✗	-	✓	?	✗	✓	✓	1
4	✓	✓	✓	✗	✓	✗	✓	✓	✓	-	✓	✓	✓	✓	✓	1
5	✓	✓	✓	✓	✓	✓	✓	✓	✓	?	✓	✓	✓	✓	✓	1
6	✓	✓	✓	✗	✓	✓	✗	✗	✗	?	?	?	✓	✓	✓	1
7	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	3
8	✓	✓	✓	✓	✓	?	✗	✗	?	?	✓	?	✓	✓	?	1
9	✓	✓	✓	✓	✓	✓	✓	✓	✓	?	?	-	✓	✓	✓	1
10	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓	2
11	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✗	✓	✓	✓	1
12	-	✓	✗	✗	✗	✗	✗	✗	✗	?	?	?	?	✗	✓	1
13	✓	✓	✓	?	✓	✗	✓	✓	✓	✗	✗	?	✓	✓	✓	3
14	✓	✓	✓	?	?	?	?	?	?	?	?	?	?	?	?	4
15	✓	✓	✓	?	✗	✓	✓	✓	✓	✓	✓	✓	?	✓	✓	4
16	✓	✓	✓	?	?	?	?	?	✓	?	?	?	?	?	?	1
17	✓	✓	✓	?	✗	?	✓	✓	✗	?	?	✓	✓	✓	✓	2

Table 4: "Full answers" collected in our survey

the second questionnaire and 10 (48%) from the first to the third questionnaire).

4.2 Analysis

Concerning goal G3 ("Evaluate if our course fosters student engagement"), results in table 2 clearly shows that our course motivates students for software design activities. Students found the proposed software design activities interesting and-or useful, they felt confident to successfully complete them, and felt challenged by them. Furthermore, the course is mainly based on project-based activities which promotes students' motivation [8]. The pedagogical approach is, for teachers, to guide students without ever imposing one solution. This approach is relevant and fosters autonomy and engagement. Regarding the differences between the two groups of students (A & D), we expected students in the group A to be more motivated skilled. However, the *M*-series and *B*-series questions show that this assumption was not founded, as results about motivation and background are similar for both groups.

Results for question D1 are good to very good, particularly for "full answers" (table 4). From figures in table 3, most of the students correctly placed software design in a software development lifecycle. This knowledge is acquired from the beginning of the course and evolves slightly over time. The practical application of this activity in a concrete project roots it for the long term.

Figures for D2, D3, and D4 in table 3 show that initial global understanding of the software design role and value is quite low but significantly improves after design activities. Moreover, for D2 and D3, in the third questionnaire, the results further improved. This indicates that some of the students learn software design value, only when the implementation is done. It should be noted that understanding that different diagrams allow to compare code-to-be-later-written without writing it is a non-obvious mental exercise that not all students are able to do. Moreover, being able to recognize properties (such as evolution, modularity or extensibility) on diagrams requires training.

As for D4 (software design drawbacks), very few students are able to mention drawbacks at the beginning of the project (18%). Although results improve in the second questionnaire (28 points), they drop again for the third one (30%). These results can be explained by the fact that all software design activities in our course are intended to make students aware of the importance of design concerning desirable software properties such as extensibility or modularity. However no insights on drawbacks are explicitly given nor time for students to reflect upon such drawbacks allocated in the course.

With respect to D5 (tools for design) in questionnaire 1, one half of the students answers the question correctly. Once software design activities are completed, this number increases to 70% and does not progress anymore. The knowledge of the tools is acquired naturally through the activities themselves and this acquisition seems long-term. For the sake of simplicity and time restrictions, we do not compel students to use a software tool for the design

activities. Teaching in a school not specialized in computer science leaves little time for tool discovering and understanding. That is why we allow (and prefer) the use of paper and pencil as a means to understanding the purpose of software design. Once this step is assimilated, we think it is easier to justify efforts in learning complex but powerful tools.

"Full answers" (table 4) clearly show an individual improvement in software design purpose understanding: 34% are correct answers, *i.e.*, answers that are always ✓ in the table and 31% represent a desirable trend. Table 4 also shows that even after the software design activities, students may improve their understanding (48% of desirable trends concern changes on answers from the first to the third questionnaire). However, this overall understanding is difficult for students even through this specially tailored course. It can also be noted that 5 "full answers" (7%) with a trend in the right direction from the first to the second questionnaire, show a non-desirable trend on answers from the second to the last one (*i.e.*, **?** or ✗). This again highlights how difficult is for students to obtain an overall understanding of the usefulness of software design.

In conclusion, for goal G1 ("Evaluate the students perception of the usefulness and pertinence of the software design phase"), students have a satisfactory overall understanding of software design activities. However, this understanding is complex to acquire and some students only master it partially. For goal G2 ("Assess the evolution, if any, of this perception along the course"), after the design activities, students clearly improve their overall understanding. The coding activities that follow the design phase allow this understanding to be rooted in a long-term way.

4.3 Threats to validity

As for internal validity, we can note:

- The number of "full answers" is quite low (17). However, we have at least as many students as they have in other similar studies (see Section 5). Nevertheless, we aim at renewing the experiment in the future with more students.
- The *D*-series questions were open and subject to interpretation. However, the teaching team agreed on an interpretation of the questions. For instance, the answer "Java" in the software design expected outcomes (*D*2) was considered as wrong, even if the eventual goal is to produce Java code.
- Questions *D*3 & *D*4 went beyond the course objectives. We hoped/expected from students a step back and an analysis from their project experience. Results are mixed.
- The repetition of questions may have led to weariness. Few late wrong responses may illustrate this.

Regarding external validity, *i.e.*, the understanding of our results in another context, we can note that:

- Even if our context is very specific (a French postgraduate non- computer science engineering school), we believe that our teaching approach may be general enough to be applied in the early stage of a computer science curriculum with similar results. Nevertheless, additional experiments would need to be conducted in order to confirm this intuition.
- *B*-series questions show a poor diversity of backgrounds. This is generally what is observed in this kind of school (French Grande École). Whether greater diversity would change results is an open question.

5 RELATED WORK

There is a plethora of studies evaluating the usefulness, advantages and disadvantages of using diverse tools, methodologies and course organizations for teaching software design and modeling in general. This is less so for studies evaluating the perception students have on the usefulness and pertinence of modeling itself and its evolution as a consequence of training.

In [3] the authors report on a survey of 47 instructors in order to capture the current status of modeling and model-driven engineering (MDE) teaching. Our course organization, content and use of modeling (focused on software engineering and design), matches some of the described courses, which places it among current practices. Instructors also report on negative aspects, such as: the students have difficulties with understanding abstraction; modeling is conceived to be too different from programming; purpose of models is unclear. One of the main objective of our study is to evaluate whether such negative aspects get alleviated over time as training takes place. In the following we focus on works that report on modeling and model-driven engineering courses which include surveys to students.

In [5], the authors present ClassCompass, a system to facilitate the evaluation of software design assignments, and evaluate how it helps students understand design principles whereas in [16] the authors discuss the rationale behind the redesign of a software design course (the refactored course is very similar to ours, focusing on a course-long main project). Both studies present the students with a questionnaire. However, they evaluate primarily the course (perceived load, difficulty, etc.) and less the perception of students w.r.t. the usefulness of modeling and design which is our focus. Some questions are similar to ours but they are less explicit and there is no tracking of the evolution of the students' opinion over time.

More focused on MDE (in the sense of rigorous models used in later stages for automatic tasks such as model evaluation, model transformation or code generation), in [4] the authors discuss the integration of MDE (including code generation) into a software design course. A survey is presented to the students with questions mainly related to usefulness of using MDE for better understanding software design. They do not take into account the evolution of answers over time. Our course does not use any automatic manipulation of models. In this sense it can be seen as a more classical use of modeling and thus complementary to their results. Close to the aforementioned work, in [14] the authors report on a new MDE course and on a set of surveys presented weekly to the students but the evolution of answers is not tracked.

More similar to our study, in [7] the authors present the students with three surveys with the objective of evaluating the evolution of their perception w.r.t. the purpose of models and its usefulness. However, their course is more focused in model transformation and the use of a number of modeling tools whereas our study focuses on the design phase of a software engineering lifecycle with no emphasis on the influence of tooling. Besides, their course is not based on a somehow large project but in smaller assignments. In that sense, our studies can be seen as complementary as well.

6 CONCLUSION AND FUTURE WORK

In this paper we have presented a study aimed at evaluating the understanding, motivation and perception of students with respect to software design. We have first described a project-based course currently taught to second-year students at IMT Atlantique and then presented three questionnaires we passed to the students enrolled to the aforementioned course in 2021. The answers to these questionnaires allowed us to analyze: 1) the students' perception of the usefulness and pertinence of the software design phase and the activity of modeling of software systems; and 2) the evolution of this perception. Results hint towards a positive evolution of the students perception on the usefulness and pertinence of software design. For some students, the design goal is difficult to understand until the effects *on* the code are observed. They realize, late, that things could have been anticipated in the design models.

In the future, we plan to continue this line of work by looking into the following directions of further work. First, we intend to continue the experiment with the future editions of the course in order to verify whether the results remain consistent and if modifications introduced to the course and/or the questionnaires positively affect the outcomes. We intend as well to refine our survey to better understand where the importance of design choices appear more evident. Is it for reification purpose? (the choice of entities that become classes) for association variants? (cardinality, level of abstraction of roles), or for method localization in classes and code distribution? Finally, we plan as well to study the influence of using different tools. E.g., would the use of a full-fledged UML editor help students better understand the usefulness of models or would it hamper it by distracting them with tools induced accidental complexity?

ACKNOWLEDGMENTS

We thank all teachers involved in teaching MAPD throughout the years for their contribution in improving this course. We also thank all students answering the surveys.

REFERENCES

[1] A. Frank Ackerman. 2014. An active learning module for an introduction to software engineering course. In *2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE T)*. IEEE, 190–191.

[2] Pierre Bourque and Richard E. Fairley (Eds.). 2014. *SWEBOK: Guide to the Software Engineering Body of Knowledge* (version 3.0 ed.). IEEE Computer Society, Los Alamitos, CA. http://www.swebok.org/

[3] Federico Ciccozzi, Michalis Famelis, Gerti Kappel, Leen Lambers, Sebastien Mosser, Richard F Paige, Alfonso Pierantonio, Arend Rensink, Rick Salay, Gabi Taentzer, et al. 2018. How do we teach modelling and model-driven engineering? A survey. In *Proceedings of the 21st ACM/IEEE international conference on model driven engineering languages and systems: Companion proceedings*. 122–129.

[4] Peter J Clarke, Yali Wu, Andrew A Allen, and Tariq M King. 2009. Experiences of teaching model-driven engineering in a software design course. In *Online Proceedings of the 5th Educators' Symposium of the MODELS Conference*. 6–14.

[5] Wesley Coelho and Gail Murphy. 2007. ClassCompass: A software design mentoring system. *Journal on Educational Resources in Computing (JERIC)* 7, 1 (2007), 2–es.

[6] Padmashree Desai and G.H. Joshi. 2012. Activity based teaching learning in software engineering - An experience. In *2012 IEEE International Conference on Engineering Education: Innovative Practices and Future Trends (AICERA)*. IEEE, 1–6. https://doi.org/10.1109/AICERA.2012.6306729

[7] Huseyin Ergin, Isaac L Walling, Kate P Rader, and D Joshua Dobbs. 2019. A study of modeling perception in a first-time modeling class. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 680–689.

[8] Scott Freeman, Sarah L. Eddy, Miles McDonough, Michelle K. Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. 2014. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences* 111, 23 (2014), 8410–8415. https://doi.org/10.1073/pnas.1319030111

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

[10] Damla Oguz and Kaya Oguz. 2019. Perspectives on the Gap Between the Software Industry and the Software Engineering Education. *IEEE Access* 7 (2019), 117527–117543. https://doi.org/10.1109/ACCESS.2019.2936660

[11] Sofia Ouhbi and Nuno Pombo. 2020. Software Engineering Education: Challenges and Perspectives. In *2020 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 202–209.

[12] Thierry Pelaccia and Rolland Viau. 2017. Motivation in medical education. *Medical Teacher* 39, 2 (2017), 136–140.

[13] Carl Adam Petri. 1962. Communication with automata. *PhD thesis* (1962).

[14] Eric J Rapos. 2018. We'll make modelers out of'em yet: introducing modeling into a curriculum. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 130–134.

[15] Pieter Van Gorp, Hans Schippers, Serge Demeyer, and Dirk Janssens. 2007. Students can get excited about Formal Methods: a model-driven course on Petri-Nets, Metamodels and Graph Grammars. In *Proc. 3rd MODELS Educators Symposium*, Vol. 7. 19–28.

[16] Ian Warren. 2005. Teaching patterns and software design. In *Proceedings of the 7th Australasian conference on Computing education-Volume 42*. 39–49.

A QUESTIONNAIRES

The following text was presented to the students at the beginning of each survey: "This survey is anonymous. Teachers will not access the answers before the publications of final course results. The purpose of this research survey is to investigate your interest in programming and design evolution during the course."

A.1 First

Before Initial Design (the beginning of the course).

B1 How did you enter IMT-Atlantique? *AST/Concours/Other*

B2 What is your interest for computer science? *No interest at all/As little as needed to succeed/Moderate/Eager*

B3 What is your self-estimated level in programming? *Beginner/Capable to write a small program (not object-oriented)/ Capable to write a small Object-Oriented program/ I'm comfortable with all kinds of Object-Oriented program*

B4 MAPDA: You are doing the DCL TAF. Please check the sentence that applies. *I didn't choose it/It was not my preferred option/It was my number one option*
MAPDD: What is your current TAF? *I didn't choose it/It was not my preferred option/It was my number one option*

B5 Concerning MAPD UE. Check the sentence that applies. *I'm willing to deeply understand/I'll do what is expected by the teachers, no more, no less/I'll study to succeed the exams*

1D1 When should software design take place during development lifecycle? *Before analysis and before realization/Before analysis and after realization/After analysis and before realization/After analysis and after realization/I do not know*

1D2 What are the expected results (artifacts) of software design? (in a few words). Answer 'I don't know' if you have really no idea.

1D3 What are the benefits of software design? (in a few words). Answer 'I don't know' if you have really no idea.

1D4 What are the drawbacks of software design? (in a few words). Answer 'I don't know' if you have really no idea.

1D5 List a few tools used during software design? (do not restrict your answers to software tools)

A.2 Second

Halfway through the course.

2D1 When should software design take place during development lifecycle? *Before analysis and before realization/Before analysis and after realization/After analysis and before realization/After analysis and after realization/I do not know*

2D2 What are the expected results (artifacts) of software design? (in a few words). Answer 'I don't know' if you have really no idea.

2D3 What are the benefits of software design? (in a few words). Answer 'I don't know' if you have really no idea.

2D4 What are the drawbacks of software design? (in a few words). Answer 'I don't know' if you have really no idea.

2D5 List a few tools used during software design? (do not restrict your answers to software tools)

M1 MAPDA: I think that software design activities are interesting and-or useful *Strongly agree/Agree/Disagree/Strongly disagree*
MAPDD: Working on software design is interesting and-or useful *Strongly agree/Agree/Disagree/Strongly disagree*

M2 I felt confident to successfully complete the software design activities of MAPD. *Strongly agree/Agree/Disagree/Strongly disagree*

M3 I felt challenged by the software design activities of MAPD. *Strongly agree/Agree/Disagree/Strongly disagree*

M4 Free comments about your motivation for the software design activities in MAPD:

A.3 Third

After the end of the course.

3D2 What are the expected results (artifacts) of software design? (in a few words). Answer 'I don't know' if you have really no idea.

3.2 Indicate your level of confidence for the previous answer. *100% sure/75% sure/50% sure/ 25% sure/0% sure*

3D3 What are the benefits of software design? (in a few words). Answer 'I don't know' if you have really no idea.

3.4 Indicate your level of confidence for the previous answer. *100% sure/75% sure/50% sure/ 25% sure/0% sure*

3D4 What are the drawbacks of software design? (in a few words). Answer 'I don't know' if you have really no idea.

3.6 Indicate your level of confidence for the previous answer. *100% sure/75% sure/50% sure/ 25% sure/0% sure*

3D5 List a few tools used during software design? (do not restrict your answers to software tools)

3D1 When should software design take place during development lifecycle? *Before analysis and before realization/Before analysis and after realization/After analysis and before realization/After analysis and after realization/I do not know*

F1 Are you convinced that software design must precede programming? *100% sure/75% sure/50% sure/ 25% sure/0% sure*

F2 When did you realize that? *I knew it before starting the course / During lectures / During the design phase of the μP / During the implementation of the μP / After the μP has been completed/Right now, thanks to the survey*