computers

MDPI

Article

Teach Programming Using Task-Driven Case Studies: Pedagogical Approach, Guidelines, and Implementation

Jaroslav Porubän [1,*], Milan Nosáľ [2], Matúš Sulír [1] and Sergej Chodarev [1,*]

[1] Department of Computers and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia; matus.sulir@tuke.sk
[2] ValeSoft, s.r.o., Vodárenská 636/3, 040 01 Košice, Slovakia; milan.nosal@gmail.com
* Correspondence: jaroslav.poruban@tuke.sk (J.P.); sergej.chodarev@tuke.sk (S.C.)

Abstract: Despite the effort invested to improve the teaching of programming, students often face problems with understanding its principles when using traditional learning approaches. This paper presents a novel teaching method for programming, combining the task-driven methodology and the case study approach. This method is called a *task-driven case study*. The case study aspect should provide a real-world context for the examples used to explain the required knowledge. The tasks guide students during the course to ensure that they will not fall into bad practices. We provide reasoning for using the combination of these two methodologies and define the essential properties of our method. Using a specific example of the Minesweeper case study from the Java technologies course, the readers are guided through the process of the case study selection, solution implementation, study guide writing, and course execution. The teachers' and students' experiences with this approach, including its advantages and potential drawbacks, are also summarized.

Keywords: computer science education; programming; software engineering; computer-aided instruction; task-driven methodology; case study approach

Citation: Porubän, J.; Nosáľ, M.; Sulír, M.; Chodarev, S. Teach Programming Using Task-Driven Case Studies: Pedagogical Approach, Guidelines, and Implementation. *Computers* **2024**, *13*, 221. https://doi.org/10.3390/computers13090221

Academic Editor: Stelios Xinogalos

Received: 19 July 2024
Revised: 29 August 2024
Accepted: 31 August 2024
Published: 5 September 2024

1. Introduction

The efficient teaching of programming is a difficult task. Often, students know the principles and processes, but they are not able to use them effectively in practice. There are even cases where professional developers have problems with applying basic principles, such as inheritance or encapsulation in object-oriented design. As a reference, the work of Hadar analyzes the reasons why professional object-oriented programmers make bad decisions although they possess the required knowledge [1]. A common reason is the lack of understanding of the context and background of a given principle. A student or a programmer has to see the problem and then come to a solution on one's own. By realizing the nature of the problem and obtaining the solution only through his own work, he will better remember the solution and even the problem to which the solution applies.

1.1. Task-Driven Teaching

The task-driven teaching method is based on the constructivism teaching theory [2]. It is based on acquiring knowledge through the experience of solving tasks. A related approach, task-centered instructional strategy, starts by presenting a series of real-world tasks that gradually form the taught content [3].

Explaining the theory in lectures is undoubtedly necessary; however, it does not provide all the understanding necessary for the application of knowledge in practice. According to Bonwell et al., active involvement of students in the learning process increases its effectiveness [4]. While people may remember the mistakes of others, they never forget their own. *Learning by doing* includes learning from one's own mistakes. Very few people can understand theory so well they will not make any mistakes when it comes to applying it in practice.

1.2. Case Studies

A case study investigates phenomena in a real-life context [5]. The case method of teaching provides a realistic situation or project to students and allows them to apply their knowledge to devise solutions to the presented problems [6]. According to Coppit, providing students with meaningful development experiences in computer science courses is essential to produce graduates who can enter the software industry and be productive [7]. Therefore, we use case studies as the basis for our method.

In general, many researchers discovered that using case studies has a positive effect on students' learning (e.g., [8,9]). However, there are not many guides available on how to design, prepare, and execute a case study in the context of computer science.

1.3. Task-Driven Case Studies

In this paper, we describe *task-driven case studies*—a specific implementation of the case study technique [6]—and share our experience with it. Our special task-driven case studies are case studies combined with the task-driven methodology. This means that the case study guides a student through multiple tasks corresponding to the learning objectives of the course. An essential part of them is study guides—written guides that describe the case study and specify tasks to be solved.

We provide guidelines on what a good task-driven case study for a course related to programming should look like and how it is prepared. These guidelines and experience may be found useful even in other types of case study implementation, i.e., without task-driven methodology, and for teaching in general.

Our previous conference article briefly described teachers' experience with our approach, analyzed the results of a survey with students, and outlined potential drawbacks [10]. In the current paper, we summarize and complement these findings, thoroughly describe a novel teaching approach devised as a combination of case studies and the task-driven approach, and provide guidelines for other educators supplemented by specific examples.

2. Background

Existing research related to task-driven case studies can be divided mainly into the topics of problem-based learning, case studies, task-driven methodology, and game-based teaching.

2.1. Problem-Based Learning

O'Grady provides an overview of some of the existing literature about problem-based learning [11]. Problem-based learning (PBL) focuses on students so they take responsibility for their learning themselves. In this aspect, PBL resembles case studies. However, in comparison with our method, PBL gives only little guidance to students. They are given a problem, and it is their responsibility to analyze and solve it. Our method guides them through a solution with tasks. Similar to challenge-based learning [12], we also draw from PBL to support students' critical thinking, self-directed learning, and long-term retention.

A more recent scoping review of PBL research articles by Ghani et al. identifies learning behaviors required for effective PBL application and can be used as a helping guide for designing better case study tasks [13].

Pérez et al. report their positive experience of introducing complex, more industrial-like projects at their university [14]. Cico et al. report that project-based learning is currently the most dominant approach to teaching software engineering [15]. Case studies enable teachers to focus more on the process of solving the project than on the end result.

2.2. Case Studies in Teaching

Many works concern the use of case studies in teaching. One of the more recent works by Grimes provides guidelines for the design of continuous case studies [16]. Continuous cases represent a type of case study that is split into segments that gradually unfold the

story represented by the case. The guidelines have many similarities with our suggestions; for example, they suggest aligning case study segments with course learning outcomes or assessing students continuously. On the other hand, the guidelines are quite generic and do not assume some specific form of task design based on the case study.

Zhang et al. explore case-based teaching of a digital image processing course [17]. They find significant improvement in several learning aspects, including students' learning interest and motivation. They also propose basic principles of case design that are quite generic.

Ouh et al. present the application of case-based teaching in a postgraduate software architecture course [18]. Instead of a large case study, they use smaller bullet cases and mini-cases mainly as a way to facilitate discussion. In our case, however, a large and more detailed case study is used and split into simpler tasks intended to be solved by students on their own.

Varma et al. present a short analysis of using case studies in software engineering courses [19]. They also provide a few general basic guidelines on how to prepare such a case study. In another work, Garg et al. provide an experimental comparison of the lecture–examination learning model and the case study method [8]. They claim that case studies help to nurture the higher-order cognitive skills of application, analysis, evaluation, and synthesis. In their later work, they discuss the advantages of using case studies as assessments [20]. In our experience, we came to quite the same conclusions, and that is the reason why case studies are used as assessments in our courses. In comparison with them, we incorporate also guidance through the case study using tasks.

A case study in a software engineering course is described also by Jia [21]. They also argue for case studies' advantages in comparison with conventional teaching approaches. Additionally, they also express the importance of the clear specification of teaching goals that we argue for in Section 4.

Burge et al. introduce a more business-oriented case study into a software engineering course [22]. In their work, they describe the case study and analyze the advantages of using it in a course. Many other works argue for using case studies and large projects as assessments. For example, experience with a case study is presented by Hilburn et al. [23]. Martin [24] and Meyer discuss the shortcomings of using introductory toy examples in comparison with large projects [25]. Coppit argues that providing students with meaningful development experiences in software engineering courses is essential to produce graduates who can enter the workforce and be productive [7]. These works identify the issue with most common academic projects so far—that projects may present a highly abstracted view of reality and students usually focus on the project outcome (product) and not the process. As a solution to these issues, the works argue for using case studies.

2.3. Task-Driven Teaching Methodology

The task-driven methodology belongs to the constructivism teaching theory. It aims to divide one standard assessment and one single task into multiple tasks. According to Yu et al., this approach is supposed to make learning more effective and faster because the knowledge could be covered naturally and the learning motivation could be maximized through full engagements [26]. In their work, Yu et al. describe their experience with a task-driven methodology.

A meta-analysis by Xie et al. shows that constructivist methods have a significant effect on improving learning in the area of mathematics compared with the traditional transmission acceptance method [27]. A similar effect can be expected in computer science as well.

Peng et al. argue that by using tasks, students' views of learning will be influenced unconsciously [28]. In the past, they used to take learning as a task; now they learn in light of the needs of the tasks, changing from passive acceptance of knowledge to active pursuit of knowledge. Peng et al. describe their specific implementation of the task-driven

methodology in one of their courses, and based on their experience, they provide some basic guidelines and advice about the application of the method.

Xue et al. describe the application of the task-driven methodology in information technology courses in detail [2]. While our work focuses more on how to devise tasks and how the tasks collaborate with case studies, Xue et al. aim more at how the task-driven methodology can be effectively executed in information technology courses.

A study by Liu et al. shows that proper usage of tasks can result in learning benefits for students [29]. Students' creative thinking is cultivated; they can work independently and at their own pace. The methodology makes hierarchical teaching more convenient through different hierarchies of tasks. Similarly, we argue the same benefit by using tasks with layered hints to provide control of the task difficulty.

Rosenberg-Kima et al. compare the effectiveness of two task-based approaches with explicit instructions in an empirical study with programming novices [3]. The whole-task strategy presents instructions in the context of the whole task, so the integration of individual components is necessary from the beginning. The part-task strategy defines distinctive components that are taught separately, while their integration is completely omitted or postponed until the tasks are finished. According to the results of this controlled experiment, students using the whole-task strategy achieved better performance and learning transfer. This empirically confirms our intuition that incremental implementation with a solution that is continually executable benefits students.

Liang et al. present a combination of task-driven and objective-oriented hierarchical education methods that they use in their Linux curriculum [30]. Objective-oriented hierarchical education enhancement allows students to solve different tasks according to their objectives in their studies. If the student's objective is to become a system administrator, he would be solving different tasks compared with a programmer. This enhancement would apply to our method as well; however, as it is expressed in Section 9, our task-driven case studies are already quite expensive to prepare. Making multiple versions according to different objectives would be very costly. A viable application is to provide different additional tasks since those are only specified as feature requirements in the problem terminology, and they do not have to be implemented by a teacher.

Dong uses a so-called graded task-driven methodology [31]. A graded task-driven methodology employs tasks graded based on their difficulty. Its purpose is to provide better students with enough challenges and worse students with tasks that they would be able to solve. These graded tasks are very similar to our idea of multilevel hints to tasks.

Birnbaum and Langmead argue that a task-driven approach is suitable for teaching programming to digital humanities students [32]. They view it as an analogy of the task-based method of teaching foreign languages, where, instead of focusing on comprehensive knowledge of grammar and vocabulary, the learning is organized around communication tasks. They suggest that this approach is suitable for students whose main area is not computer science. While our method is focused on computer science students, its principles are in line with the suggestions provided by Birnbaum and Langmead, including the important role of the case study domain, or incremental development. Therefore, it can be adapted to other study programs as well.

In comparison with all of the aforementioned works about task-driven methodology, our task-driven case studies differ in combining the task-driven methodology with case studies. In this combination, we wanted to give students a complex real-world environment in a case study combined with proper guidance through tasks to teach them best practices.

2.4. Games in Teaching and Automatic Feedback

In many of our courses, we implement games as case studies. If possible, we recommend taking this approach, as games are a proven tool for motivating students and encouraging their engagement in the learning process [33,34]. Zhan et al. review and analyze multiple recent studies that show the positive effects of gamification on the learning process [35]. Papadakis documents another study with evidence of the positive motivational effects of

games on students in learning programming [36]. Jordaan provides evidence that even board games can improve students' engagement in learning programming [37].

Tay et al. examine and describe game design elements needed to inhibit students' motivation and engagement in the learning process [38]. We refer to them for some useful guidelines on the design of a game case study. Useful information on this topic can be also found in [39,40].

Cavalcanti et al. summarize a review of recent works in providing automatic feedback to students in an online environment [41]. As suggested at the end of Section 8.1, task-driven case studies open space for utilizing automatic feedback and obtaining its benefits.

3. Task-Driven Case Studies' Pedagogical Framework

As was explained earlier, task-driven case studies are a teaching approach that combines case studies with the task-driven methodology. The approach consists of the following three main components:

1. **Case study**, a realistic project providing an opportunity to showcase most of the learned topics.
2. **Tasks** defined in the context of the selected case study and corresponding to the learning objectives of the course.
3. **Written study guides**, explaining the case study, leading students through the tasks, and providing enough context and explanation.

3.1. Reasons to Use Tasks

By using the task-driven approach, it is possible to change students' view of learning [28,30]. Instead of perceiving learning as a task, they start learning because of the tasks. This view changes their attitude from passively accepting knowledge to pursuing it for the needs of the tasks. Such a learning model better mimics industrial practice.

A task-driven case study is built upon relatively fine-grained tasks. These tasks define what a student should do to continue in the case study. More specifically, programming courses contain implementation tasks, defining what needs to be implemented. They guide students through the process of solving the case study.

The following are the three main reasons why we use tasks:

1. We want to make sure that students are active during lessons. Both the teacher and the students are reminded that practical lessons should revolve around the students working and not the teacher lecturing.
2. We want the student to use best practices while working. Finer-grained tasks help us to keep the student on the right track. In learning, the process is more important than the result.
3. Instead of only showing the process and the students repeating after the teacher, we use task-driven case studies to make them try the process by themselves. Then it is harder to forget about the details.

In our method, case studies are combined with the task-driven methodology, incorporating tasks to guide students to active participation in practical lessons. The whole case study with the tasks is presented to students using written study guides.

3.2. Role of Written Study Guides

The written study guides are an integral part of our method. They contain all the important information for solving the case study or, in general, to fulfill the goals of the practical lessons. The following are the three main reasons why we recommend preparing written study guides for task-driven case studies or even for task-driven teaching in general:

- A written form of the guide allows students to work at their own speed. If there is something unclear, they can get back to it anytime.
- Students can continue working at home without any impedance. If a student is unable to come to the lesson, he misses only individual consultations with a teacher. To a

large degree, the lesson can be simulated by the study guide. This proved to be very useful during COVID restrictions.
- A study guide can be shared between universities or learning facilities.

Study guides substitute the teacher in explaining the case study (or the contents and tasks of the lesson) to the students. The teacher can focus on explaining only the more problematic aspects of the case study and individually helping the students with their problems.

The tasks represent the main component of the study guides—each lesson should have at least one task. However, they are not the sole content of them. Besides the tasks, the study guide for each lesson provides the students with a problem definition and explanation of the problem context, objectives of the corresponding lesson, background, reasoning, further reading, etc. These concepts will be explained in the rest of the paper.

3.3. Course and Case Study Relationship

To prepare a good case study for a course, its author has to realize the connection between the course itself and the case study. Figure 1 demonstrates such a relationship.

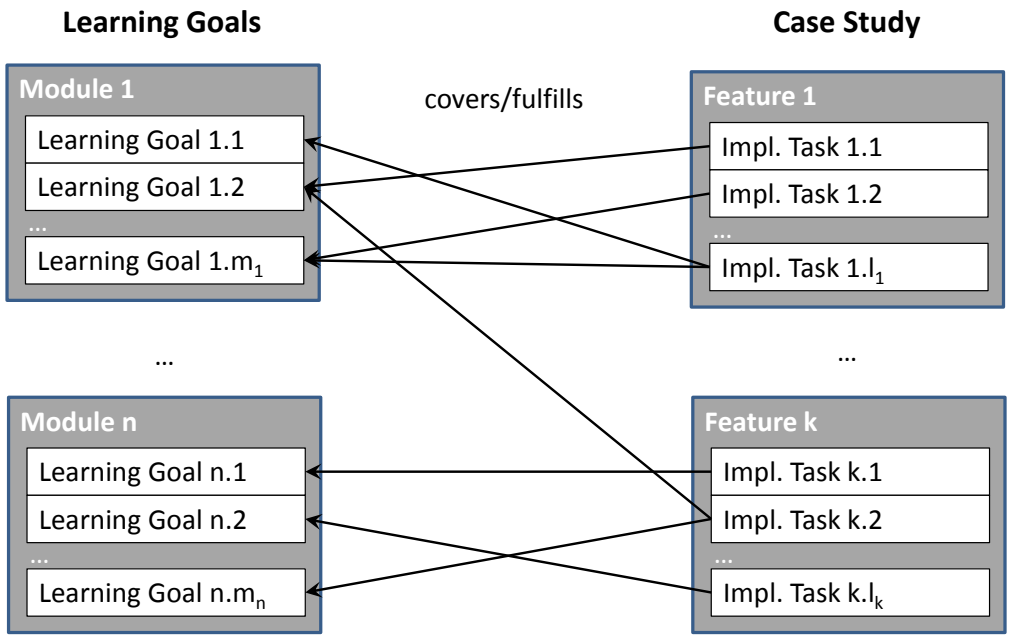


Figure 1. The relationship between goals and tasks in case studies.

Each course has a set of learning goals (displayed on the left side of Figure 1). For example, the Programming Basics course could have goals such as "understand the meaning of conditional statements" and "know how to use iterations". Learning goals are the core of the course. The course is usually divided into modules, where each module contains a set of learning goals for one lesson of the course.

Everything in the course should revolve around these goals, so the case study has to concentrate on them too. One very important issue is mapping the learning goals to the case study tasks.

A case study solution (right side of Figure 1) consists of multiple functional features of an implemented product. Each of these features is implemented by solving a sequence of implementation tasks. A task is a specification of some implementation goal that has to be achieved by a student to continue in the case study, e.g., "perform empty game field generation" or "shuffle the field items randomly". Each implementation task covers and fulfills one or more learning goals. Each goal can be covered by multiple tasks. If there is a

goal that is not covered by any task, we risk that the goal material will not be exercised in full and therefore not be comprehended well.

According to our experience, it is a good practice to map each feature completely into one module only so that the feature can be runnable after the lesson. However, this is not always possible, but as we argue later, it is useful to have an executable solution after each module, and mapping like this can help to achieve that.

3.4. Task-Driven Case Study Lifecycle

For preparing a good case study, it is useful to understand the case study lifecycle. We have identified a lifecycle of the task-driven case study as it is presented in Figure 2.

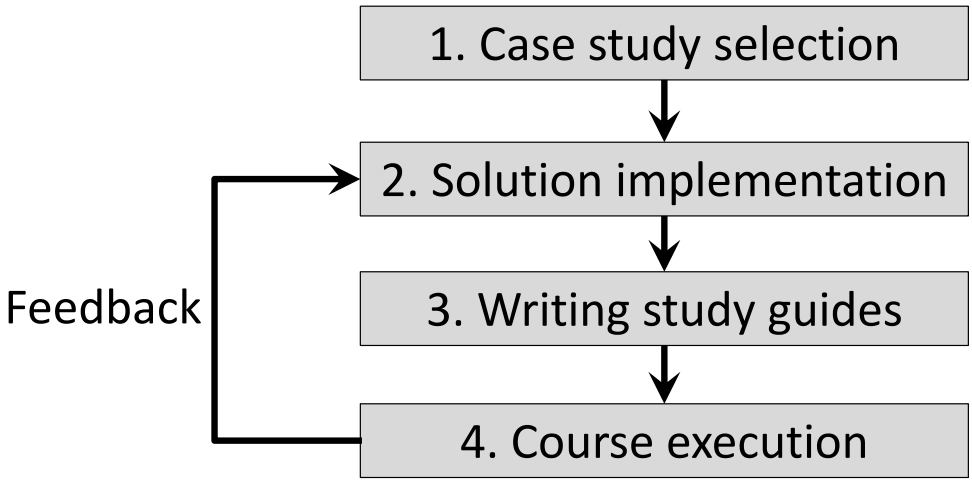


Figure 2. The task-driven case study lifecycle.

The lifecycle starts with the case study selection phase. In this phase, the authors need to find a suitable theme and project that is appropriate for the course. The selected project should offer enough space to fulfill the learning goals. After the selection phase, we suggest starting with the design and implementation of a solution to refine our understanding of the details and find potential problems. The solution implementation phase is followed by the writing study guides phase. After the study guides are finished, it is time to run the course. During the execution, it is essential to collect feedback from the students and subsequently include the suggestions and observations in the study materials and update the implemented solution.

3.5. Guidelines

In this paper, we use our experience to extract guidelines that apply to our approach or even to case studies or the task-driven methodology in general. The guidelines will be backed with examples from our most mature case study—the Minesweeper from the Java technologies course.

The guidelines are structured based on the task-driven case study lifecycle phases. We discuss each of them in detail in the following sections. Here, however, we provide only a summary of them in the form of lists as an overview or a checklist for course authors.

Selection of the Case Study

- Know the learning goals.
- Select a well-known domain.
- Find an interesting topic.
- Keep the project in industrial quality.
- Show the whole process.

- Prepare for incremental implementation.
- Support individual approach.

Solution Implementation

- Implement code first.
- Divide code into goals.
- Evaluate goals' coverage.
- Find goal dependencies.
- Do the review.

Writing Study Guide

- Always tell objectives.
- Describe the current state and expected increment.
- Show context and reasoning.
- Specify tasks clearly and precisely.
- Refine tasks with comments.
- Add supplementary tasks.
- Provide further reading.
- Use suitable software support.
- Verify continuity and consistency of the guide.
- Review the guide.
- Start with a minimal version and improve it.

Course Execution

- Monitor progress continuously.
- Favor individual achievements.
- Examine understanding.
- Get feedback from the "battlefield".
- Update the case study.

4. Selection of the Case Study

In the *case study selection* phase, the author should find a project that would be appropriate for a case study. A good subject of a case study can have a positive effect on students' motivation, satisfaction, and learning efficiency. The following are the guidelines that stemmed from our experience.

4.1. Know the Learning Goals

As a first step, it is really important to identify the learning goals for students in the course. Clear, sound, and specific learning goals are a basis for preparing a course, and they should be prepared upfront. Goals help to select a proper case study project. According to our experience, without having explicitly stated goals, it is hard to prepare tasks that would cover all the required topics in the course.

Example: For our case, the Minesweeper case study for the Java technologies courses, we have written down a list of learning goals that would lead to understanding the Java language and all the fundamental Java technologies. The following list presents an excerpt of the goals of the Java technologies course:

- Students are familiar with Java language basics.
- Students can create a project and a class in the NetBeans IDE.
- Students understand the role of interfaces in object-oriented programming.
- Students can implement existing interfaces in Java.
- Students can create their own interfaces in the Java language.
- Students are familiar with the role and types of collections in the Java language.
- Students can work with the generic `ArrayList` collection.

- Students can use `String` class methods for working with strings.
- Etc.

These goals are specific and can be used later to check whether the implementation of the case study covers all the necessary topics in the course.

4.2. Select a Well-Known Domain

We experienced that a proper domain of the case study project should be a domain well known to the students. When the students are supposed to learn to program in a Java language, they should not be bothered with understanding a domain (unless it was stated as a learning goal). A well-known domain helps to save time and energy in favor of learning the course topics.

Example: You can imagine that, for example, the problem of luggage transport at an airport is not a very good choice. Most of the students never had to (or will never have to) deal with that problem, and some of them probably never even traveled by plane. That is the reason why we picked a Minesweeper game. Thanks to the Microsoft Windows operating system, it is one of the most known games. We could barely find a student in the Java technologies course who had never played Minesweeper.

4.3. Find an Interesting Topic

Motivating students is one of the most challenging parts of education. Without students willing to learn, teachers may use state-of-the-art methodologies and approaches, but they will hardly obtain the expected results. Using the words of Mingins et al. [42], "The course should be fun!".

We have to impress students with their assignments, ideally making them work with impressive programs, like those that handle graphics. Teaching can capitalize on this "wow effect". "Trying to get students excited is pedagogy, not demagogy" [25].

In this context, it is important to know the target group. Our experience proved that the easiest way to get today's students' attention is to make them implement their own game—we use game-based learning [43]. Many of our students claim that they came to study Informatics only because they wanted to learn how to make games.

Example: That is why we chose the Minesweeper game. In some of the other courses, games are used too. For example, we used the N-Puzzle game in the .NET programming course and Alien Breed clone in the object-oriented programming course.

4.4. Keep the Project in Industrial Quality

We suggest implementing a project that is of industrial quality. Students need to feel the touch of the real world, of the industry. Many times, students' motivation can be killed if they have to solve problems that seem to be too trivial or unrealistic.

Another benefit of an industrial-quality product is its motivational aspect. Students can flaunt their solutions to their friends and family. In this context, the domain also plays a role. For example, a game has a greater impression on non-IT people than a console or desktop application.

The projects should also implement best industrial practices.

Example: The solution to our Minesweeper case study is comparable to the professional Minesweeper game (see Figure 3). Students can see that they are able to implement a program comparable to the industry.

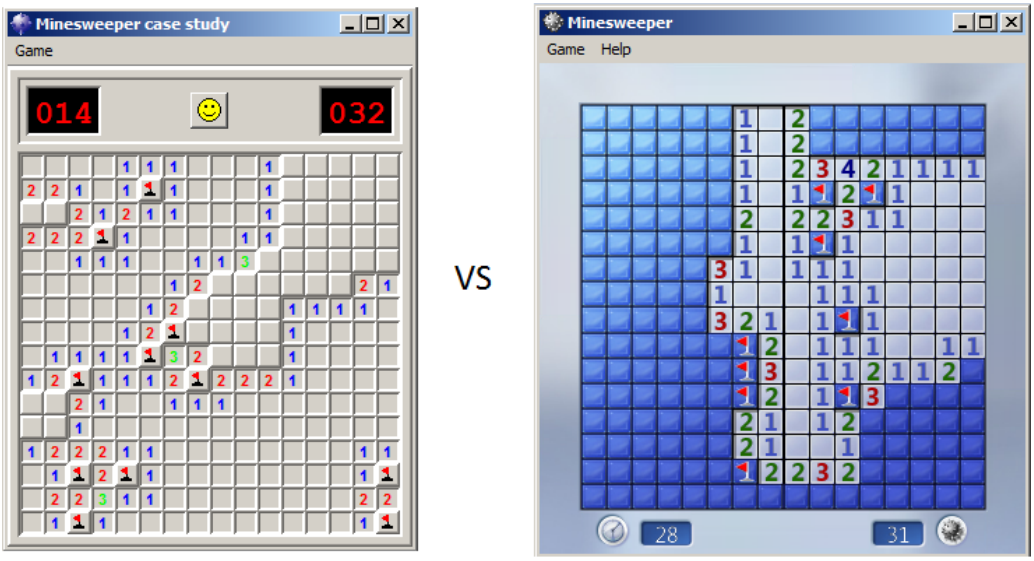


Figure 3. Our case study solution (**left**) compared with Microsoft Minesweeper (**right**).

For example, our Minesweeper case study has a modular design that strictly separates individual user interfaces: the console (textual) and the graphical user interface.

4.5. Show the Whole Process

We want to teach students the whole process of implementing a good software system. Showing only partial solutions and expecting that they will somehow understand the glue that put these together is naive. Garg et al. [8] talk about receiving feedback that students felt that software engineering is a theoretical and unexciting subject and is perhaps of very little use in the future. The reason for this feedback may be found in the lack of experience related to a project [19,44]. We, on the other hand, had very good feedback on our case studies that guided them through the whole process of implementing the project (see Section 8).

Students usually focus on the project outcome (product) and not on the process of the project implementation. However, the process is a part of the course during which they should learn how to apply theoretical knowledge. It is not enough to show them what they should apply. As Garg et al. say, the competence of a student in applying skills or the synthesis of learning in real-world scenarios should be established from a case study carefully designed by an experienced professional [20]. Throughout solving the case study, students have to be guided to use best practices, not only to implement a solution that will work. Instead of merely showing them the best techniques and expecting them to somehow become programming masters, the case study shows them how a masterpiece is done (in an ideal case, when the case study really covers a masterpiece). The case study shows them how the work and reasoning go through the whole process.

The tasks should not focus only on the implementation phase itself. To abstract the problem from implementation details, in some courses, students first write pseudocode in natural language and draw diagrams (flowcharts, UML, etc.) as necessary. In some courses, the first one or two weeks are dedicated to the analysis, while in other ones, the beginning of each lab can be focused on the analysis of the problem. The details depend on a specific course and are a part of the study guide.

To be able to show the whole process, the project size has to be picked very carefully. Students should implement a reasonable part of the project themselves, not only a small part. We want them to see the creation of a comprehensive solution (especially important when teaching a programming paradigm, e.g., object-oriented programming). Therefore, a final solution cannot be a large application or just its small part. It has to be something smaller yet comprehensive—not only, for instance, a sorting algorithm.

Example: In our Minesweeper case study, we chose the Minesweeper game because it is small enough to be implemented by a student in the time span of the course, and yet the result is a whole game that is comparable with industrial Minesweeper.

4.6. Prepare for Incremental Implementation

In our experience, it is very good for students' motivation if, after each lesson (module), the partial solution is runnable. Students can immediately see the results and test their implementation. If the students work for 10 weeks and see no result, they may lose the courage and motivation to continue. "*What if I have made some error in the beginning and the whole implementation is now incorrect?*" Seeing the results gives them the feeling that it is going the right way. Moreover, it gives them satisfaction with the work they had performed during that particular lesson. Therefore, we suggest choosing a case study that supports such incremental implementation.

Example: In the Minesweeper case study, the first case study lesson introduces the Minesweeper game. We are designing the game core together with students using standard UML notations. During this, they are led to our source code skeleton implementing best practices from object-oriented programming. In the next lesson, students implement the game logic—the game field and all its behavior. So far, their solution is not playable. However, we do not wait anymore, and in the third lesson, they start implementing a simple console user interface that would be able to present the current state of the field. Although, after the third lesson, they cannot play the game, they can already run the game and see whether their field generation works as it should. Since that moment, they are always able to run the game and see the increment they add during each particular lesson.

4.7. Support Individual Approach

Most of the previous advice aimed to keep students on the right track when implementing the case study. We argued that we have to guide them so they can see how a professional solution is created. However, students' creativity should be nurtured too. Therefore, we recommend selecting a project that would provide students with enough space for their individuality. Students should follow the guide in creating the main program, but then, they must have the opportunity to enrich it with their own ideas and special features.

Example: In our Minesweeper case study, we suggest that students implement some additional features that the main project does not have. An example may be implementing the support for a simultaneous click of both mouse buttons to open also adjacent tiles and not merely the one that the mouse points to. We also support their own ideas on how to make the game even more interesting.

5. Solution Implementation

After a good subject for the case study was selected, in the next step, the teacher should implement the case study himself—the *solution implementation* phase. It helps the teacher to get familiar with the case study and the study solution, with all its details and problems. The sample solution will verify if the selection was really appropriate.

5.1. Implement Code First

If the teachers start writing a study guide and tasks without first trying to implement them, they risk that the students will stumble upon some unseen issues that may complicate the learning process. We always implement the case study first ourselves. It helps us to see every detail of the solution, make the best design decisions, and provide sound reasoning about them to the students. There should not be any open problem in the main project that the teachers would not be aware of and to which they would not have an answer. The implementation-first approach also helps the teachers to avoid guiding the students the wrong way. It is better if they find out upfront that there is some problem than if the students find it out during the course.

Example: We implemented our own Minesweeper game before giving the case study to students. Before using the case study approach in the Java technologies course, the Minesweeper game was one of the examples that we used to show the students more complex examples implemented in Java. Then we decided to prepare a case study and reworked the old solution to incorporate all the topics of the course and to reflect our best knowledge.

5.2. Divide Code into Goals

It is recommended that the teacher divide the source code into learning goals they achieve. In the *case study selection* phase of the case study creation process, we recommended preparing a list of learning goals. These goals specify what a student has to learn in the course. Now the source code of the implementation should be matched to these goals. We recommend doing it in two steps. First, the teacher identifies implementation objectives needed to implement the case study and maps the code to them. The implementation objectives represent the tasks, such as students have to implement the generation of the game field and students have to implement the presentation of the game field. Second, the teacher maps the implementation objectives to learning goals. For example, the implementation objective "students have to implement the presentation of the game field" would be mapped to the learning goal "students understand and know how to use data streams".

We recommend explicitly marking the division in the code as well—-by using either comments or preferably source code annotations (attribute-oriented programming [45]). Having the source code of the case study explicitly annotated by its implementation objectives (and through them to learning goals) helps with the maintainability and evolvability of the case study.

Example: To explicitly record implementation objectives in the code of the Minesweeper case study, TODO comments marked with the task identifiers were used. Each task represented an implementation objective. We started using TODO comments because of the tool support—see the automatically generated list of TODO comments in an IDE in Figure 4. However, current IDEs have better support for source code annotations since they are first-class citizens of the language. The following code is an excerpt showing a TODO comment marking the `getColumnCount()` method as a part of the task identified as "getters" in the second module (lesson) of the course.

```
//TODO: Task 2 - getters
/**
* Returns column count of the field.
* @return column count.
*/
public int getColumnCount() { ...
```

An alternative using source code annotations (e.g., a custom `@Task` annotation) is presented in this second listing:

```
/**
* Returns column count of the field.
* @return column count.
*/
@Task(module = "02", id = "getters")
public int getColumnCount() { ...
```

In this example, the `getters` task can be easily identified in the source code. The task itself then needs to be mapped to a learning goal. In our example, it would be the goal of the learning principles of implementing the Java Beans convention. To illustrate the indirect mapping through tasks, Figure 5 shows relationships between the "Swing User Interface" feature implementation tasks of the Minesweeper game and the learning goals in the Java technologies course that spans multiple case study modules. The diagram is an instantiation of the diagram presented in Figure 1. It shows how implementation

tasks relate to learning goals. The coverage of tasks in the code is represented by code annotations (or legacy TODO comments).

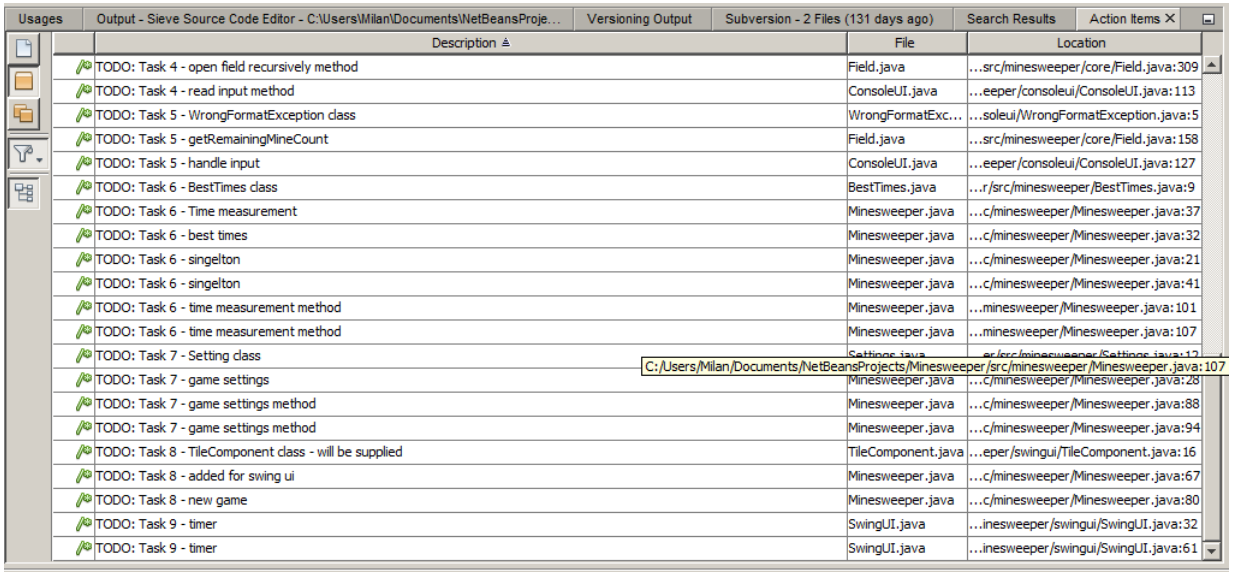


Figure 4. The projection of code division to goals using TODO comments in Minesweeper.

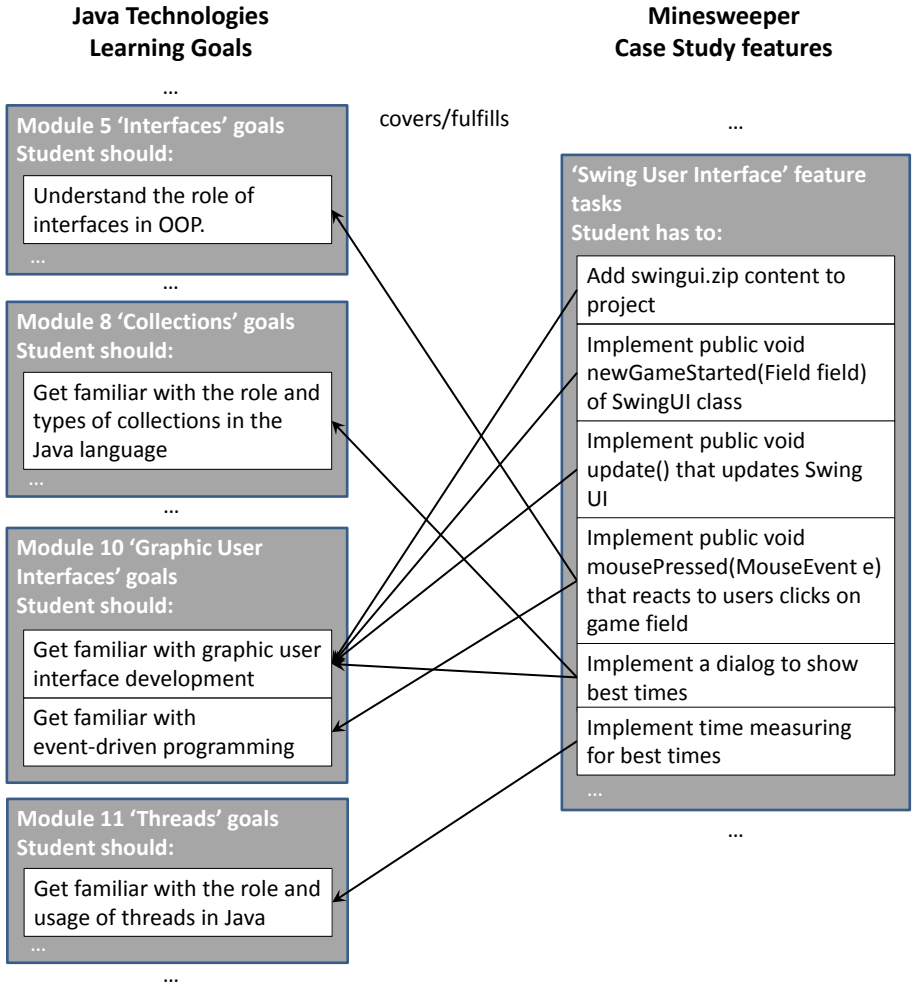


Figure 5. Relation between implementation tasks and learning goals in the Minesweeper case study.

5.3. Evaluate Goals' Coverage

It is important to verify that every goal is addressed in the source code. In this process, IDE tools can be used to check the explicit code division according to implementation tasks.

It is also crucial to go through every goal and consider the difficulty of its tasks too. Some goals may not be that important, and they may include a lot of code and, therefore, a lot of work for students, while other important goals may be covered only by simple implementation problems. The idea is to balance the code with the importance of its goal.

Each learning goal has to be covered in the case study—not vice versa—not every piece of code has to be relevant to learning goals. In such cases, these pieces of source code can be given to students ready-made. If students cannot learn anything relevant to the course from implementing that code, it is usually a waste of time to make them do it.

Example: A simple help in checking the code coverage is, for example, the Action Items window in the NetBeans IDE (previously called the Tasks window). In Figure 4, there is a screenshot showing the list of tasks currently present in the Minesweeper teacher's solution. It is easier to check the goals this way since the goals' source code can be scattered throughout the whole project.

In the Minesweeper case study, the students are given the code skeleton of the game core—it is the result of object-oriented design and does not address any specific Java technology. Thus, its creation would fit in the OOP (object-oriented programming) course, but not in the Java technologies course.

5.4. Find Goal Dependencies

Learning goals and their tasks need to be divided to form a list of modules. One module defines one lesson. Modules should be balanced so that all the lessons are approximately of the same difficulty. The order of the tasks for a lesson is very important. In our experience, the best time to prepare the order of the lessons and their tasks is in the *solution implementation* phase of the case study creation. Usually, the order will follow the order in which the implementation is made by the teacher.

The most important aspect in the ordering of goals is their implementation dependencies. If the code of one task cannot be implemented unless a code of some other task is present, then the other task has to be fulfilled first.

For the sake of having continuity, it is also useful to verify whether the tasks for one goal are not scattered too much in the implementation process. For example, when the students are exercising interfaces, they should not be forced into other topics until they are finished with interfaces.

One should also consider the principle of incremental implementation from the *case study selection* phase. If there is no implementation dependency, the goals should be ordered in a way that each learning module will end with a runnable partial solution. From the motivational viewpoint, this aspect has a higher priority than the continuity aspect.

After preparing the order of the goals, we recommend aligning the lectures with the case study (practical lessons). It is really important to ensure that each topic exercised in the case study was already discussed in the lectures.

Example: In the Java technologies course, the lectures are aligned with case study modules so that the students always have the theoretical knowledge that is needed for the current module of the case study. The alignment is depicted in Figure 6.

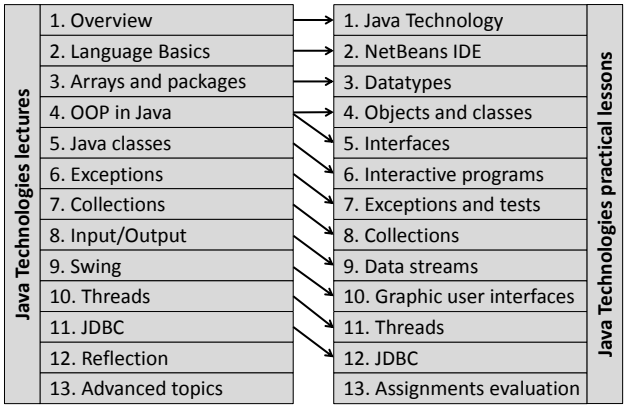


Figure 6. Alignment of lectures and practical lessons in the Java technologies course.

5.5. Do the Review

After finishing the teachers' sample solution, the best practice is to get it reviewed by other professionals. This may mean other teachers who are teaching the course or even experts from the industry. The feedback from them can significantly improve the case study.

6. Writing Study Guide

We recommend preparing written study guides instead of merely presenting the problem and the case orally for a few reasons that have already been mentioned in Section 3. First, the case study is usually non-trivial, and therefore, it is better to have its written specification, so the students can get back to the parts they did not understand anytime they want. Second, they can work at home, having a copy of a study guide or accessing it online. Third, the quality of case study specification does not differ even when there are multiple teachers with different knowledge and skills. Fourth, this way, a case study can be easily shared between universities or other teaching facilities.

Writing a study guide for a case study follows the *solution implementation* phase. However, the teacher does not have to wait until the completion of the implementation of the whole solution. He can implement a part of the solution for the first lesson, and then write a study guide for the first lesson, and only then continue with the next one.

To better mimic the process instead of only a specification, we suggest that the study guide for each module should consist of a sequence of steps. Each step would bring a student closer to the finish. Steps consist of the problem context for the step and *tasks* that have to be implemented.

Example: To get a better notion of how such a study guide looks, a specific example of one lesson from the Java technologies course can be found at https://kurzy.kpi.fei.tuke.sk/tjava-en/student/06.html (accessed on 30 August 2024).

6.1. Always Tell Objectives

Students should know what is their objective for the current lesson. The best way to begin a study guide for a lesson is to start it by listing the objectives. This can include learning goals, but also the implementation objectives that specify project features or their parts that should be implemented during the lesson. Sometimes the students are afraid of some important topics because they consider them complex and difficult. By telling the students more and providing concrete information, we want to show them that there is nothing to be afraid of.

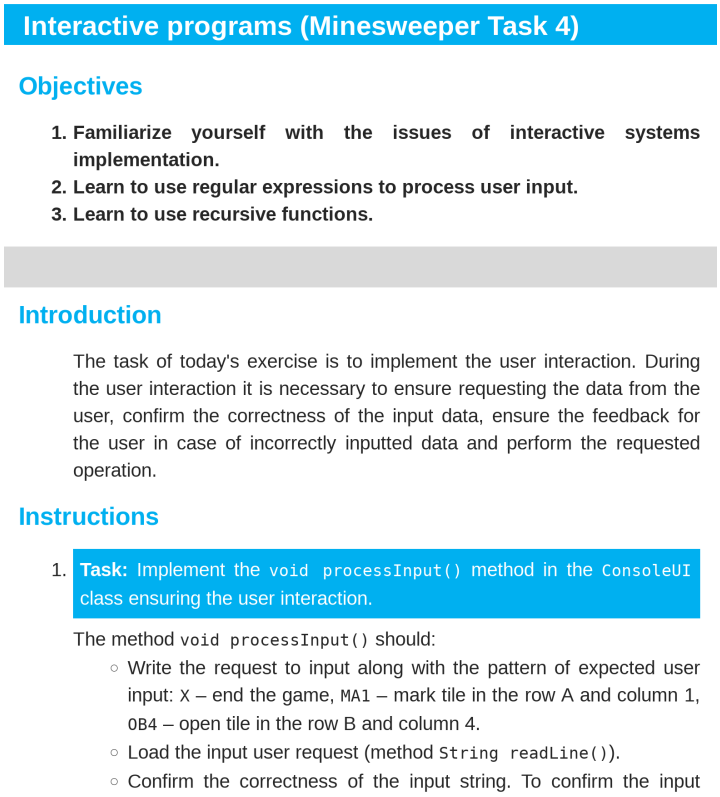
After the lesson, students can also verify that they did not miss any important point by checking whether they fulfilled and understood each goal/objective.

Last but not least, we found it very useful to link the learning goals in the study guide to the materials used in lectures. This way, students can easily get to the relevant information from the lectures if it is needed.

Example: Examples of learning goals are presented in Section 4 when we discussed making a list of the learning goals for the case study. However, there are also implementation objectives. These are specific to each case study. For example, in the following list, there are some implementation objectives from the Minesweeper case study.

- Students have to implement the generation of the game field.
- Students have to implement the presentation of the game field.
- Students have to implement a time-measuring feature for the game.
- Students have to implement the settings feature.
- Students have to implement a graphical user interface in Swing.
- Etc.

In Figure 7, there is a beginning of a study guide page for one of the lessons of the Java technologies course. Each goal (objective) is mapped to one or more steps in the instructions part, which contains explanations for students along with implementation tasks. In the example, the first task fulfills goal numbers 1 and 2 since it implements user interaction and uses regular expressions to achieve this.



Interactive programs (Minesweeper Task 4)

Objectives

1. **Familiarize yourself with the issues of interactive systems implementation.**
2. **Learn to use regular expressions to process user input.**
3. **Learn to use recursive functions.**

Introduction

The task of today's exercise is to implement the user interaction. During the user interaction it is necessary to ensure requesting the data from the user, confirm the correctness of the input data, ensure the feedback for the user in case of incorrectly inputted data and perform the requested operation.

Instructions

1. **Task:** Implement the `void processInput()` method in the `ConsoleUI` class ensuring the user interaction.

   The method `void processInput()` should:
   - Write the request to input along with the pattern of expected user input: `X` – end the game, `MA1` – mark tile in the row A and column 1, `OB4` – open tile in the row B and column 4.
   - Load the input user request (method `String readLine()`).
   - Confirm the correctness of the input string. To confirm the input

Figure 7. An excerpt from a study guide demonstrating the relationship between goals and tasks.

6.2. Describe the Current State and Expected Increment

After listing the objectives, a good start is to recapitulate the current state of the solution (the state in which the implementation should be after previous lessons) and to show the desired increment to it. Of course, it is not always necessary to recapitulate the complete current state of the solution. Many times, an abstraction (e.g., a UML class diagram) focused on the current problem will suffice.

Example: We use class diagrams in each module to show the difference in the program structure before and after finishing the current lesson. In Figure 8, there is a class diagram of the Minesweeper case study in the fifth module. The yellow classes represent the current state, and the red ones are the increment for the fifth lesson. This helps the students to understand how the implementation goals will be projected into the program structure.

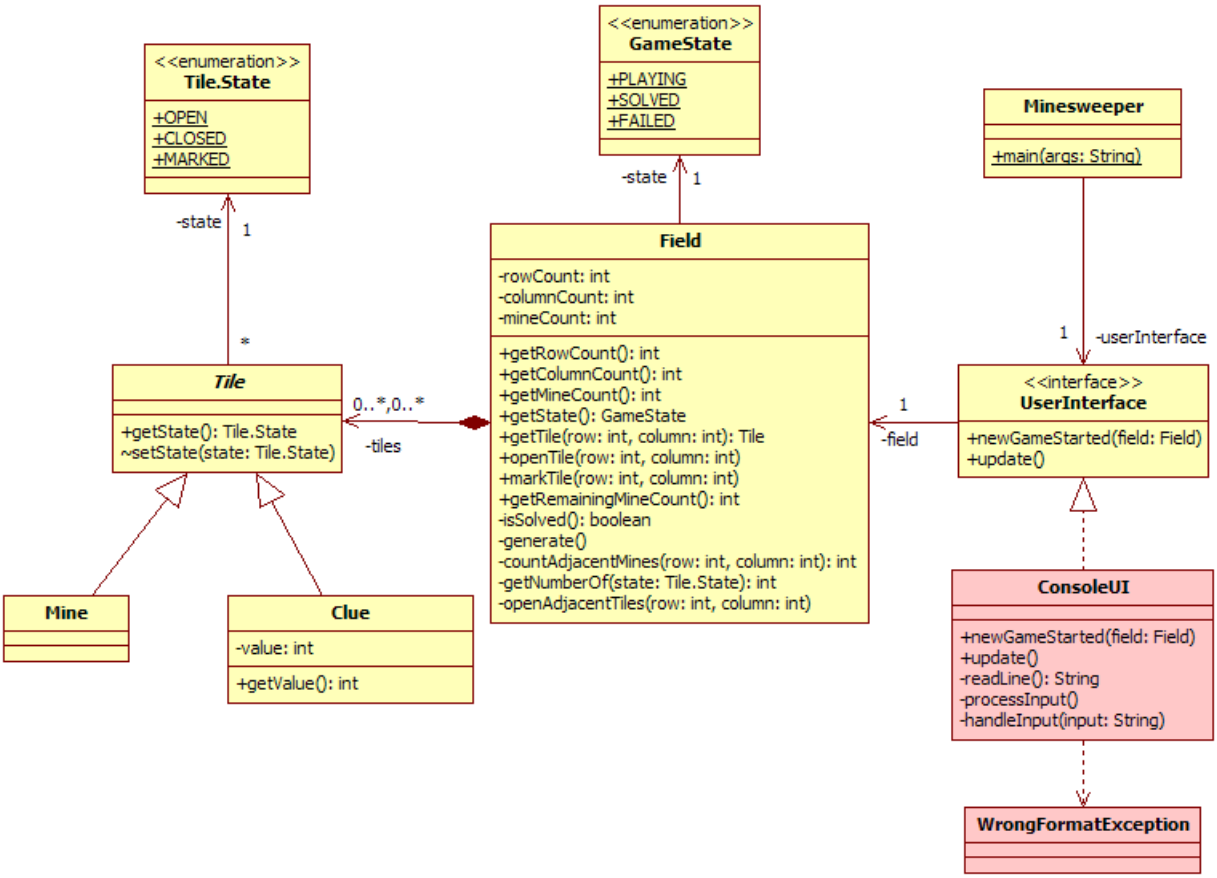


Figure 8. A snapshot of the Minesweeper class diagram in the fifth module.

6.3. Show Context and Reasoning

It would not be a case study if it did not provide a clear context and reasoning. In the study guide, there has to be a part that should explain to students the reasoning and motivation for the work they will be doing in the current step of the lesson. It is also the specification that can be used to verify the correctness of the student's increment implementation.

Example: In the Minesweeper case study, each step starts with an explanation of the current situation, the problem context, and the reasoning behind it. This part of the step is intertwined with the tasks that lead to solving the problem—each step of the module starts with the context and reasoning, and after that, the tasks follow.

6.4. Specify Tasks Clearly and Precisely

Each step provides some context to the current problem, motivation, and reasoning. A step consists of this context and of tasks that guide students in solving the problem.

The most important and hardest part of the case study creation is the clear and precise specification of the tasks. Task specifications should not be too long. If a task is too complex to be explained in a few sentences, then it should be divided into subtasks. Additionally, multiple shorter tasks are used to control the students' implementation process so they will not fall into bad practices.

The level of abstraction for task specification depends on the context. Usually, at the beginning of the course, the tasks are described more thoroughly. For instance, precise pseudocode is provided and the students can simply rewrite it to the given programming

language. Near the end of the course, tasks tend to be described more briefly since students have already gained experience.

Tasks for a single lesson should be balanced. They do not have to be of the same difficulty, but it is important to keep the difficulty generally non-decreasing.

Example: Following is an example of a task from our Minesweeper case study that tells the student to implement a method that will be a part of the Minesweeper marking tiles feature. This is one of the simpler tasks from earlier lessons of this introductory Java course.

```
Task (id = markTile)

Implement the void markTile(int row, int column) method in the Field
class. This method allows marking/unmarking tiles specified by the
row and column. In~case the tile is closed (Tile.CLOSED), its state
will be marked (the state will change to Tile.MARKED). If~a tile is
marked (Tile.MARKED), its state will be changed to closed (Tile.
CLOSED). Rows and columns are numbered from 0.
```

When the students would get to this task, they should already have a case study code skeleton with a bit of their code. In the `Field` class, students would find the following `markTile(int row, int column)` method stub:

```
/**
* Marks tile at specified indices.
* @param row row number
* @param column column number
*/
public void markTile(int row, int column) {
    throw new UnsupportedOperationException("Method markTile not yet
    implemented");
}
```

The following is an example of a possible solution to this task:

```
public void markTile(int row, int column) {
    final Tile tile = tiles[row][column];
    if (tile.getState() == Tile.State.CLOSED) {
        tile.setState(Tile.State.MARKED);
    } else if (tile.getState() == Tile.State.MARKED) {
        tile.setState(Tile.State.CLOSED);
    }
}
```

This task is quite simple. However, with the case study progress, the tasks are growing in difficulty and provide less guidance, as the students are expected to build upon the already-passed tasks. For example, one of the latest tasks in the course requires the user to handle best scores persistence through a database connection, as follows:

```
In the BestTimes class define a private void insertToDB(PlayerTime
playerTime) method that will store a PlayTime object in the database.
```

The students are also provided with a couple of general notes about creating database connections in Java and about prepared statements with links to official documentation, but otherwise, the students have to work on their own to come up with a correct implementation.

6.5. Refine Tasks with Comments

We suggest adding comments with hints to tasks to refine their difficulty. The tasks need to be challenging even for above-average students. However, in such a case, the students that are below the average could become overwhelmed. To keep the tasks challenging for the better students while not making it too hard for the others, we recommend using hints in comments that would help with solving the tasks.

Students can start solving the task without looking at the comments at all (our study guides are rendered as HTML pages, and therefore, the hints can be hidden by default). Then, if a student finds the task too difficult, he can take a look at hints. There may be even levels of the hints that can be later taken into consideration when the solution is graded. For example, without using comments, a student would obtain a full number of the points, and then looking at each comment (hint) would be penalized by subtracting some points. Multilevel hints would allow for the finer-grained difficulty of tasks.

Example: The following is an example of a hint for the "markTile" task that was presented above. It advises using the implementation of the `openTile(int row, int column)` method for inspiration. The `openTile(int row, int column)` method does a very similar job and is provided to students in the Minesweeper code skeleton at the beginning of the course. Most of the students should not have any serious problems in solving the task without this hint. However, below-average students or programming novices may struggle with it, and this hint should refer them to the right direction even before they will need to ask the teacher for help.

```
When implementing the void markTile(int row, int column) method, you
can use the implementation of void openTile(int row, int column)
method as an inspiration.
```

6.6. Add Supplementary Tasks

The aforementioned tasks and comments should cover all the learning goals. After that, the students can face open problems in additional features with the experience they obtained from the main project. To support individuality, some supplementary tasks are added to each lesson, recommending features that are not covered by the standard tasks. Additionally, students are also asked to come up with their own ideas about additional features.

Supplementary tasks provide a space to try out the learned principles in a less restrictive environment. They are specified in the problem terminology. Only a feature is requested, but we do not add any information about implementation. Students have to solve the problem themselves now without our guidance.

Example: In the Minesweeper case study, the implementation of new features is suggested, such as support for a new state in marking tile—using the question mark to signify that the user is not sure whether there is a mine or not.

6.7. Provide Further Reading

A module can be concluded with a list of links to additional literature and resources. There can be links to blogs, tutorials, documentation, books, or even scientific papers—anything that might be interesting for the students who want to know more.

Example: In the Java technologies course, we recommend the book *Head First Java* for further reading. For the advanced study, *Effective Java* is recommended. For particular topics, students are usually provided with links to specialized tutorials and blogs.

6.8. Use Suitable Software Support

In the final stage, the study guide should be available online for students in some format, usually as a website. Since it is already time-consuming to devise the study guide itself, the technical requirements should not further complicate this process and distract the teacher from focusing on writing the content. Instead of hand-crafting HTML code or designing the guide in desktop publishing software, the teachers can utilize static website generators that automatically produce websites from simple text documents written in formats such as Markdown or reStructuredText [46].

Example: At our university, we developed a specialized document generator for task-driven case study guides. This program transforms a set of Markdown files enriched with metadata representing the goals, tasks, etc. into a static website and possibly other formats, such as PDF. Utilizing custom software for this task also opens possibilities to check for inconsistencies, such as goals without any corresponding tasks.

6.9. Verify Continuity and Consistency of the Guide

After writing the study guides, they need to be checked for inconsistencies. All the tasks have to be correct, and they cannot contradict each other. Additionally, the coverage of the solution has to be checked. If a student follows the study guide and solves the main tasks (not additional), his solution has to be complete. If some implementation detail is forgotten, a student will probably face frustration—-he followed the guide, and yet the solution is not complete or, in a worse case, not even working. Explicit task marking in the source code can help to ensure that all the tasks in the code are stated in the study guide too. Unmarked code that does not belong to any task has to be given to students ready-made.

6.10. Review the Guide

The study guide should be reviewed too. However, for reviewing the guides, we use colleagues from different courses and older students. Their task is to try solving the case study and give us feedback about the form of the guide, the used terms, the style of writing, and so on. Older students can tell us whether the study guides are clear and easily understandable. With colleagues from different courses, we can unify our terminology.

The older students can provide the teacher with feedback about the overall difficulty of the case study. They can warn the teacher that the case study is too difficult for the target audience (since they are not so "distant" from it as the teacher's peers). This can be dealt with by providing students with more ready-made source code (a skeleton of the program, or some complex method), hints, or more elaborated task specifications.

6.11. Start with a Minimal Version and Improve It

This point applies both to the solution implementation and to study guide writing. We recommend first preparing a simple version of the source code and instructions that the students can already use and then gradually extending and improving them. This is similar to the concept of a minimum viable product in software engineering.

Regarding specific time frames, they are highly dependent on the specific course, teachers' experience with the subject, and many other factors. We managed to produce a minimal version of materials for one course in as little as two weeks of very intensive work. An upper limit is usually thirteen weeks, i.e., one semester. Therefore, it is possible to develop the materials continuously if the interconnection between individual lessons is carefully analyzed upfront.

A whole course can be implemented by one person or a small group of teachers. In the latter case, close cooperation and synchronization are necessary. In our experience, if the whole course consists of one large case study, two teachers are the upper practical limit. If multiple smaller case studies are provided, the number of teachers can be larger.

7. Course Execution

When the study guides are reviewed and ready, the case study can be put to use in a course. Students are given the guides, and they have to solve the case study. The case study is divided into modules to match the course lessons. We recommend beginning each lesson by introducing the current objectives and the problem context. Then students should be left to work on their own on the solution by solving tasks. The study guide should not substitute teachers but rather help them to get more time to approach students individually.

7.1. Monitor Progress Continuously

Based on our experience and literature [47], it is necessary to monitor students' progress continuously. They need to keep up with the plan that is determined by the study guides. When they start getting behind the plan, it usually gets only worse, and then often comes a moment when they feel overwhelmed and give up (or try to cheat by obtaining the implementation from their peers). Each module should be designed to be solvable in the time of the lesson, plus some homework time. Doing two or three modules at the same time might stress them so much that they will give up. However, we had also an opposite experience when good students finished the whole case study in a week or two because they found it so interesting.

Using multiple partial deadlines might help to keep the students working continuously. Requiring to deliver partial solutions after each lesson also mimics the industry, so it has the benefit of modeling the practice.

This monitoring also helps teachers to find out about students' errors as early as possible and to help them with misunderstood topics.

7.2. Favor Individual Achievements

To obtain better grades, students should be encouraged to exercise the skills that they obtained by working on the supplementary tasks. Performing only the mandatory tasks should be a minimum for passing the course. If a student wants a better grade, he should be proactive. This grading motivates students to try to apply the gained skills on their own.

Better students might find a case study too easy. In those cases, we let them work on their own projects, but first, they have to finish a case study to see the best practices (they are usually able to finish it in a few weeks instead of the whole semester). As a reward for this additional work, these students do not have to take the standard exam, but they rather have to prove their understanding of the principles by showing their application in their own projects.

Example: In the Java technologies course, we reward finishing the Minesweeper case study without any supplementary tasks by half of the maximum points available. The rest of the points can be obtained through additional tasks and custom features.

7.3. Examine Understanding

By finishing the case study, students exercised all the taught principles in practice. That does not necessarily mean that they have understood the topics and techniques. Therefore, the understanding has to be evaluated during the exams. Since the students were working on a comprehensive project the whole course, it is possible to assign them tasks concerning this project without risking that they will lose too much time comprehending the tasks' context.

Example: For our exams, an incomplete modified version of the Minesweeper solution is used. Students get to implement some missing part where they have to use the skills they should have acquired from the case study. The tasks are not the same as those in the case study, but they are similar in character. The following is an example of the exam task:

```
Implement the processInput() method that will process input from the
console using regular expressions. After~the implementation, the~game
 should be playable. After~each printing of the field, the~game would
 require typing some input. The~input is in the same format as in the
 case study: (X) EXIT, (MA1) MARK, (OB4) OPEN.
```

7.4. Get Feedback from the "Battlefield"

Reviews from the "training grounds" (colleagues, experts from industry, and older students) are very important, but these reviews will not discover all the issues with the case study. The best feedback is from the target audience.

Example: One of the important feedback entries we received from the students in the Java technologies course was the information that the task of implementing a method that counts adjacent mines to a tile is too difficult for many students (algorithmically). It did not look so difficult to us or our colleagues.

7.5. Update the Case Study

Feedback should be used to increase the quality of the case study and to keep it up-to-date. This update means going back to the *solution implementation* phase, modifying the teacher's solution, and then updating the guides as well. We highly recommend keeping the teacher's solution up-to-date with the guides, not only to update the study guides. An example of the benefits this brings is the fact that if a new teacher joins the course, it is much easier for him to get familiar with the case study.

Example: After finding out that the implementation of counting the adjacent mines is too difficult for the students, we decided to provide a hint to the task. If necessary, we could also include a note in the teachers' version of the study guide about discussing the algorithm with students.

8. Experience

We have been successfully using case studies in our courses for over 18 years. Our first task-driven case study started in 2006. Task-driven case studies have been used in several different courses at our university: the Minesweeper case study in the Java technologies course, N-Puzzle in the .NET platform programming, the Share Me case study in the distributed programming course, Karel the Robot in the C programming course, and others. Our point of view as teachers is presented here and supplemented by a summary of a survey that we conducted with our object-oriented programming students.

8.1. Teachers' View

Students are more active. Before the introduction of task-driven case studies, the teacher spent a large part of the practical labs explaining the instructions to the students. This resulted in less time allocated to the students' practical experience. However, with task-driven case studies, the whole lesson is focused on solving tasks.

Individual progress speed is a consequence of having the study guide with a detailed description of the case study and its tasks. Students follow the guide and ask a teacher for advice when a problem with the realization of the case study arises. The guide is accessible to students not only during the lessons but also during the rest of the semester. A student who is slower than average can continue with the study at home—and vice versa—an ingenious student does not have to wait for the teacher's explanation and can progress faster. This way, we can nurture better students while not forgetting about the rest.

The teacher has more time for an individual approach. Since, in an ideal case, the study guide gives the students all the necessary information, it partially substitutes the role of the teacher. This does not remove the need for a teacher though. The teachers act rather as consultants, and they can spend the time saved by the guide discussing the specific problems the individual students face.

The guide narrows the gap in the expertise of teachers. Because of the large number of students in our courses, there are usually multiple teachers per course. Since each teacher has a different set of qualities and abilities, we could not guarantee exactly the same conditions for everyone. The written guide gives all the students the same basic environment.

The case study presents the whole process. Since the case study is a whole project, not only an explanatory example, we can incorporate professional methods and tools commonly used in industry. This offers the students the feeling of methods and tools used for the design, testing, versioning, etc.

Students see showpieces, not toy examples, and therefore, we can show them the best practices. The case study and tasks lead them through our reasoning, so they understand and follow our design decisions (which were verified by multiple reviews).

The teacher monitors the students' progress, so they will not fail to deliver the assignment because of procrastination. Since the case study is divided into relatively balanced modules according to the number of practical lessons in the course, the teacher can easily determine the portion of the project that is already done. Instead of vague and ambiguous statements such as "I am working on a database layer", they are asked about which task they already fulfilled.

Teachers can share study guides with other universities and facilities. If the case study is good, other educators might be interested in using the same case study in their courses. Thanks to written study guides and pre-made teachers' solutions, the task-driven case study is highly reusable.

Task-driven case studies open space for tools in the process of teaching. All of the students have to work on the same project that has a certain main skeleton that they have to follow. Standard, non-supplementary tasks can be therefore checked automatically by a tool; in case of standard mistakes and errors, the feedback can be provided to students; etc.

Although the method has many benefits and is worth applying in education, we are aware of its imperfection. The drawbacks of the method are outlined in Section 9.

8.2. Students' View

To evaluate the advantages and shortcomings of our approach from the students' point of view, we conducted a survey with students that were using a task-driven case study during one of their courses.

8.2.1. Objective

Our main goal was to answer the following research question: how do students perceive the task-driven case study approach when applied to their course?

We used a mixed-method research approach and gathered both quantitative and qualitative data, with a main focus on the quantitative part.

8.2.2. Method

An online questionnaire was constructed using Google Forms. It consisted of 13 single-choice, 1 multiple-choice, and 3 open-ended questions. For a list of questions, see Table 1.

The questionnaire was sent to all second-year computer science bachelor's degree students who took the object-oriented programming course, where the task-driven case study was used. They also had previous experience with this method from the programming course. In the given semester, a portion of the students were using a more mature case study called Text Game, specifically an adventure with a textual command-based interface. The rest of the students used the new Alien Breed case study, which is a game with 2D isometric graphics. Participation in the survey was voluntary; we received 112 responses out of 180 invitations.

For single-choice and multiple-choice questions, absolute and relative numbers of responses for each answer were computed using spreadsheets. The responses to the free-form questions were read by one of the authors, gradually grouped into categories, and then summarized textually.

Table 1. Survey questions and results.

#	Question	Answer	No.	%
1	Do you like learning by implementing a game?	**Yes**	103	92
		No, I would rather implement something else	9	8
2	How do you like the implemented game?	Poor (is it still a game?)	0	0
		Below average (I would never play it)	18	16
		Average	56	50
		Above average (I would definitely try it)	36	32
		Excellent (from now on, I will play only this game)	2	2
3	Would you show your game to your friend or a family member?	**Yes**	87	78
		No	25	22
4	In practical lessons, you prefer to implement:	**One large project (as in Alien Breed or Text Game)**	84	75
		Multiple simple independent tasks	28	25
5	Do you think you understood programming principles better by implementing one large project?	**Yes**	91	81
		No	21	19
6	From the point of view of assignment organization, you prefer:	**Study guide that leads me through the process**	97	87
		To get the assignment in the beginning and to solve it on my own	15	13
7	Did you have problems with dependencies between lessons, i.e., that you had to solve a previous lesson to be able to continue?	Yes	48	43
		No	64	57
8	Was the difficulty of the tasks in the case study balanced?	Yes	48	43
		No, some were too easy and some too difficult	64	57
9	Were the tasks too easy?	Yes, most of the time, I just needed to repeat what was written	3	3
		No, I usually had to think more about it	109	97
10	Were the tasks described clearly enough?	**Yes, usually I understood the task without the help of the teacher**	57	51
		No, often I had to ask the teacher or colleagues for help	55	49
11	Would you like learning with study guides for a case study in future courses? *	**Yes**	103	92
		No	8	7
12	Do you think that working with a study guide:	Limits me because I cannot do what I want	22	20
		Does not limit me; I still have enough space for my individuality	90	80
13	When did you implement the tasks for a given lesson?	I programmed mostly before the lesson	12	11
		I programmed during the lesson	24	21
		I programmed after the lesson	76	68
14	Which properties of studying with study guides for a case study do you consider most important (choose max. 3)?	I implemented a large project	69	62
		I implemented a game	34	30
		Thanks to the online study guide, I could work at my own speed	81	72
		I had a study guide that lad me to good practices	55	49
		I worked incrementally, but the game was always playable	42	38
15	What did you like about practical lessons in the OOP course?			
16	What did you dislike about practical lessons in the OOP course?			
17	What would you change or improve about practical lessons in the OOP course?			

* One of the respondents did not choose any response for this question.

8.2.3. Results

The results for individual questions are displayed in Table 1. In the rest of the section, the results concerning the individual aspects related to task-driven case studies will be summarized.

Motivation, particularly intrinsic motivation, is an important aspect of learning efficiency [48]. Although the task-driven case study approach is not limited to games, they represent an interesting topic for a case study. Indeed, 92% of the students stated that they like learning by implementing a game in contrast with other types of applications, as can be seen in question 1. In question 3, the majority of the participants (78%) responded that they would show their game to a friend or family members, which could be one of the motivational factors to implement the application with their best effort.

Question 3 also emphasizes the need to implement a project in **industrial quality and in a well-known domain**. Here, we analyzed also responses for the Text Game and Alien Breed case studies individually. While, for the former, only 67% of the students would present their solution to other people, for Alien Breed, this number reached 81%. Text Games are less known nowadays, while Alien Breed represents a genre of graphical action games that many people recognize. Students also believe that graphical applications are more relevant to their future jobs in the industry. Similar results were achieved in question 2, where the students also preferred Alien Breed: 41% perceived their creation as above average or excellent, and only 11% in the case of Text Game.

The importance of **showing the whole process** and **not using toy examples** was confirmed by questions 4 and 5. Three-quarters of the students preferred one large project instead of multiple small independent tasks. A total of 81% thought that they understood programming principles better by using this approach.

Questions 6 and 11 were focused on **task-oriented study guides**. A vast majority of the students (87%) preferred guides leading them through the process of project implementation in contrast with receiving only instructions at the beginning. Furthermore, 92% of the participants would like to use task-driven study guides in future courses.

Since **dependencies between lessons** are present in task-oriented case studies, students should finish previous modules to continue with the next one. For 43% of the students, this presented a problem sometimes, while 57% did not encounter any difficulties with lesson dependencies. This should be taken into account by the instructors, e.g., by reminding the students of this fact periodically and by continuously checking the progress.

In questions 8 to 10, the **quality of tasks and their descriptions** was evaluated. More than half (57%) of the students did not consider the tasks well balanced. However, this is not a crucial property of task design, although it can make solving them more manageable. Only 3% thought the tasks were too easy, and they just needed to read the provided study guides leading them through the whole process and repeat the instructions in their own projects without considerable thinking effort. According to half of the students (51%), the instructions were clear enough to proceed without help, while the rest had to ask the teacher or colleagues additional questions. We think that the clarity of tasks could be improved by providing multilevel hints (additional hints with gradually more detailed information) in the future.

We were also interested in whether the study guides did not limit the students' **creativity**. According to the results of question 12, the majority of the students (80%) thought that the guide still left enough space for their individuality. This could be attributed also to the possibility of own extensions and enhancements in addition to the mandatory tasks.

The importance of **individual progress speed** was evaluated in question 13. Since the study guides with complete instructions are available online, the students can program not only during the lesson (21%) but also before it (11%) or after it (68%). The last option was popular because students often prefer to work in their home environment with their own equipment, and they are frequently unable to fully finish all tasks during the time of the lesson itself.

Summing up, the task-driven case study method is popular among students, with a vast majority desiring to apply it also in their future courses. Care must be taken mainly to describe the tasks clearly and provide additional hints if necessary.

9. Potential Drawbacks

There are also some potential drawbacks to using task-driven case studies. Along with the drawbacks that we experienced, our approaches to decrease their effect are presented.

Case study costs—one of the most significant reasons against using a task-driven case study is the costs of preparing and evolving such a case study. Preparing a good task-driven case study is time-consuming.

One of the problems causing the increase in costs is the relationship between the teacher's solution and the study guides. Working on the two resources simultaneously is more complicated due to synchronization. We have proposed a solution using source code annotations to define tasks in the teacher's solution and to generate the contents of the study guides from these annotations. More about this work can be found in [49].

Task difficulty—balancing the difficulty of the tasks is really hard. They cannot be too challenging so that all students would be able to solve them. On the other hand, the tasks cannot be so easy that the case study would not become a simple tutorial. Our experience is that sometimes a good difficulty can be achieved only after trying a case study during a course and obtaining feedback from students (in Section 8, our experience with the find adjacent mines method was mentioned).

In Section 6, we mentioned comments with hints that can help you in dealing with this issue.

Creativity—in our approach, all students work on the same case study, and the whole process is controlled by the study guides and teachers. From this view, it may look like the creativity of students is impeded.

We deal with this issue by trying to find a balanced ratio of mandatory tasks and additional tasks.

Strong dependencies—the lessons usually strongly depend on each other. If a student misses a lesson for some reason (e.g., sickness), he cannot continue without solving the missed lesson. It is not possible to start solving the current lesson unless the source code from the previous lesson is finished. In our experience, when students decide to take a longer break, it is really hard for them to get working again.

To keep them working continuously through the course, their progress at every lesson is monitored, and when we see that they are behind, we try to find out why and help them catch up.

Instant execution—in some case studies, it is not possible to achieve program executability after every lesson. In our experience, a runnable version after each lesson acts as a motivational factor. Having an executable program allows them to check for possible mistakes and differences between the requirements and their solutions.

When there is no way to make some lesson content executable, we suggest providing students at least with unit tests that would give them feedback about whether their solution to the lesson's task is correct. Besides the advantage of executability, unit tests mimic the real industrial process.

To deal with this issue, there is an alternative to the incremental approach. The differential approach gives a whole solution to students from the very beginning. However, the solution is provided as a black-box library. Throughout the lessons, they are given new libraries, each one missing more from the implementation. The students' task is to implement the missing part. This way, the solution is runnable after each lesson. However, this approach has its issues too. Here, part of the motivational factor is lost since, after each lesson, they do not see anything new—the solution is always the same, and changing is merely a ratio of their implementation to the library.

In practice, we use a rather combined approach. Students are building the solution incrementally, but to speed up the parts that would make a lesson inexecutable, students are provided with some code written by a teacher. When one builds a real-scale application, he cooperates on it with other developers, and the teacher's source code mimics this.

Plagiarism—plagiarism and cheating are not exclusive problems of the case studies. However, since there is usually only one case study per course, it is easier for students

to obtain a solution from older students or their peers. It would be possible to prepare multiple case studies to teach the same goals, but yet one good case study is an expensive and challenging task for a teacher.

Furthermore, case studies sometimes make it more difficult to decide what constitutes plagiarism. Students' solutions use the same skeleton, and some methods are so simple that it is not easy to write completely new implementations even when a student works alone.

To some degree, it is possible to apply traditional preventive and repressive measures for plagiarism, such as more frequent checking of the students' progress or automated similarity checks.

10. Conclusions

While both case studies and the task-driven methodology are well-known teaching approaches, their combination is our contribution. This paper represents an introduction to task-driven case studies. It identifies its lifecycle and shows the proper relationship between the case study and the goals of a course. In our experience, case studies combined with the task-driven methodology have proven themselves a worthwhile approach to teaching. We have backed our experience also with a survey with students, who appreciated also the study guides that are an integral part of the method.

An important part of the paper is guidelines for preparing and executing a task-driven case study. These guidelines are extracted from our experience with using task-driven case studies over multiple years. They should be useful for teachers that start with our method, but many of them can be used also in the context of teaching in general.

Although we were able to devise an effective approach to teaching, there is still a lot of space for improvement. Section 9 mentioned multiple problems with task-driven case studies. We have already made some progress in the case study costs, which was published in one of our previous papers [49]. However, there is still an important problem with the definition of tasks, which was identified by the survey too. In this matter, we plan to experiment with multilevel hints. Additional quantitative evaluation is also one of the tasks for future research.

Author Contributions: Conceptualization, J.P. and M.N.; methodology, M.N.; software, J.P., M.N., and S.C.; validation, M.S. and M.N.; formal analysis, M.S.; investigation, M.N.; resources, J.P.; data curation, M.N.; writing—original draft preparation, J.P., M.N., M.S., and S.C.; writing—review and editing, J.P., M.N., M.S., and S.C.; visualization, M.N.; supervision, J.P.; project administration, J.P.; funding acquisition, J.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by VEGA No. 1/0630/22 Lowering Programmers' Cognitive Load Using Context-Dependent Dialogs.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The anonymized results of a survey with students are available in the public repository at https://doi.org/10.17605/OSF.IO/WT7C4 (accessed on 30 August 2024).

Conflicts of Interest: Author Milan Nosáľ was employed by the company ValeSoft, s.r.o. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Hadar, I. When intuition and logic clash: The case of the object-oriented paradigm. *Sci. Comput. Program.* **2013**, *78*, 1407–1426. [CrossRef]
2. Xue, J.; Zhang, L. Application of task-driven approach in information technology education. In Proceedings of the International Conference on Electrical and Control Engineering 2011, 2011, ICECE 2011, Yichang, China, 6–18 September 2011; pp. 2024–2026. [CrossRef]
3. Rosenberg-Kima, R.B.; Merrill, M.D.; Baylor, A.L.; Johnson, T.E. Explicit instruction in the context of whole-tasks: The effectiveness of the task-centered instructional strategy in computer science education. *Educ. Technol. Res. Dev.* **2022**, *70*, 1627–1655. [CrossRef]

4. Bonwell, C.C.; Eison, J.A. *Active Learning: Creating Excitement in the Classroom*; ASHE-ERIC Higher Education Report No. 1; School of Education and Human Development, The George Washington University: Washington, DC, USA, 1991.
5. Anwar, S. Use of engineering case studies to teach associate degree electrical engineering technology students. In Proceedings of the 31st Annual Frontiers in Education Conference, Washington, DC, USA, 10–13 October 2001; Volume 3, pp. 8–10. [CrossRef]
6. Nilson, L.B. *Teaching at Its Best: A Research-Based Resource for College Instructors*; John Wiley & Sons: Hoboken, NJ, USA, 2010.
7. Coppit, D. Implementing large projects in software engineering courses. *Comput. Sci. Educ.* **2006**, *16*, 53–73. [CrossRef]
8. Garg, K.; Varma, V. A Study of the Effectiveness of Case Study Approach in Software Engineering Education. In Proceedings of the 20th Conference on Software Engineering Education Training, Dublin, Ireland, 3–5 July 2007; CSEET '07; pp. 309–316. [CrossRef]
9. Daun, M.; Salmon, A.; Tenbergen, B.; Weyer, T.; Pohl, K. Industrial case studies in graduate requirements engineering courses: The impact on student motivation. In Proceedings of the 2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE&T), Klagenfurt, Austria, 23–25 April 2014; pp. 3–12. [CrossRef]
10. Porubän, J.; Nosál', M. Practical experience with task-driven case studies. In Proceedings of the 2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 4–5 December 2014; pp. 367–372. [CrossRef]
11. O'Grady, M.J. Practical Problem-Based Learning in Computing Education. *Trans. Comput. Educ.* **2012**, *12*, 10:1–10:16. [CrossRef]
12. Leijon, M.; Gudmundsson, P.; Staaf, P.; Christersson, C. Challenge based learning in higher education– A systematic literature review. *Innov. Educ. Teach. Int.* **2021**, *59*, 1–10. [CrossRef]
13. Abdul Ghani, A.; Fuad, A.; Yusoff, M.S.B.; Hadie, S.N.H. Effective Learning Behavior in Problem-Based Learning: A Scoping Review. *Med Sci. Educ.* **2021**, *31*, 1199–1211. [CrossRef] [PubMed]
14. Pérez, B.; Rubio, A.L. A Project-Based Learning Approach for Enhancing Learning Skills and Motivation in Software Engineering. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education, New York, NY, USA, 11–14 March 2020; SIGCSE '20; pp. 309–315. [CrossRef]
15. Cico, O.; Jaccheri, L.; Nguyen-Duc, A.; Zhang, H. Exploring the intersection between software industry and Software Engineering education - A systematic mapping of Software Engineering Trends. *J. Syst. Softw.* **2021**, *172*, 110736. [CrossRef]
16. Grimes, M.W. The Continuous Case Study: Designing a Unique Assessment of Student Learning. *Int. J. Teach. Learn. High. Educ.* **2019**, *31*, 139–146.
17. Zhang, X.; Zhang, B.; Zhang, F. Student-Centered Case-Based Teaching and Online–Offline Case Discussion in Postgraduate Courses of Computer Science. *Int. J. Educ. Technol. High. Educ.* **2023**, *20*, 6. [CrossRef]
18. Ouh, E.L.; Irawan, Y. Applying Case-Based Learning for a Postgraduate Software Architecture Course. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, Aberdeen Scotland, UK, 15–17 July 2019; pp. 457–463. [CrossRef]
19. Varma, V.; Garg, K. Case studies: The potential teaching instruments for software engineering education. In Proceedings of the Fifth International Conference on Quality Software, 2005, QSIC 2005, Melbourne, VIC, Australia, 19–20 September 2005; pp. 279–284. [CrossRef]
20. Garg, K.; Varma, V. Case Studies as Assessment Tools in Software Engineering Classrooms. In Proceedings of the 22nd Conference on Software Engineering Education and Training, Hyderabad, India, 17–20 February 2009; CSEET '09; pp. 8–11. [CrossRef]
21. Jia, Y. Improving software engineering courses with case study approach. In Proceedings of the 5th International Conference on Computer Science and Education 2010, Hefei, China, 24–27 August 2010; ICCSE 2010; pp. 1633–1636. [CrossRef]
22. Burge, J.; Troy, D. Rising to the Challenge: Using Business-Oriented Case Studies in Software Engineering Education. In Proceedings of the 19th Conference on Software Engineering Education and Training, Turtle Bay, HI, USA, 19–21 April 2006; pp. 43–50. [CrossRef]
23. Hilburn, T.; Towhidnejad, M.; Nangia, S.; Shen, L. A Case Study Project for Software Engineering Education. In Proceedings of the 36th Annual Frontiers in Education Conference, San Diego, CA, USA, 27–31 October 2006; pp. 1–5. [CrossRef]
24. Martin, F. Toy Projects Considered Harmful. *Commun. ACM* **2006**, *49*, 113–116. [CrossRef]
25. Meyer, B. Software engineering in the academy. *Computer* **2001**, *34*, 28–35. [CrossRef]
26. Yu, D.; Wang, Q. Task-Driven Method in Practical Teaching of Software Engineering. In Proceedings of the Third Pacific-Asia Conference on Circuits, Communications and System 2011, Wuhan, China, 17–18 July 2011; PACCS 2011; pp. 1–3. [CrossRef]
27. Xie, C.; Wang, M.; Hu, H. Effects of Constructivist and Transmission Instructional Models on Mathematics Achievement in Mainland China: A Meta-Analysis. *Front. Psychol.* **2018**, *9*, 1923. [CrossRef] [PubMed]
28. Peng, W.; Jingjing, X. The implementation and harvests of task-driven in basic computer education at university. In Proceedings of the International Conference on E-Health Networking, Digital Ecosystems and Technologies 2010, Shenzhen, China, 17–18 April 2010; EDT 2010; Volume 2, pp. 311–314. [CrossRef]
29. Liu, H.H.; Su, Y.S. Effects of Using Task-Driven Classroom Teaching on Students' Learning Attitudes and Learning Effectiveness in an Information Technology Course. *Sustainability* **2018**, *10*, 3957. [CrossRef]
30. Liang, L.; Deng, X.; Liu, Q. Task-driven and objective-oriented hierarchical education method: A case study in Linux curriculum. In Proceedings of the IEEE International Symposium on IT in Medicine and Education 2008, Xiamen, China, 12–14 December 2008; ITME 2008; pp. 316–318. [CrossRef]

31. Dong, Y. A Graded Task-driven Methodology for Computer Science Education. In Proceedings of the Second International Workshop on Education Technology and Computer Science 2010, Wuhan, China, 6–7 March 2010; ETCS 2010; Volume 3, pp. 654–656. [CrossRef]
32. Birnbaum, D.J.; Langmead, A. Task-Driven Programming Pedagogy in the Digital Humanities. In *New Directions for Computing Education*; Fee, S.B., Holland-Minkley, A.M., Lombardi, T.E., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 63–85. [CrossRef]
33. Martinez, L.; Gimenes, M.; Lambert, E. Entertainment Video Games for Academic Learning: A Systematic Review. *J. Educ. Comput. Res.* **2022**, *60*, 1083–1109. [CrossRef]
34. Mayer, R.E. Computer Games in Education. *Annu. Rev. Psychol.* **2019**, *70*, 531–549. [CrossRef] [PubMed]
35. Zhan, Z.; He, L.; Tong, Y.; Liang, X.; Guo, S.; Lan, X. The effectiveness of gamification in programming education: Evidence from a meta-analysis. *Comput. Educ. Artif. Intell.* **2022**, *3*, 100096. [CrossRef]
36. Papadakis, S. Evaluating a Game-Development Approach to Teach Introductory Programming Concepts in Secondary Education. *Int. J. Technol. Enhanc. Learn.* **2020**, *12*, 127–145. [CrossRef]
37. Jordaan, D.B. Board Games in the Computer Science Class to Improve Students' Knowledge of the Java Programming Language: A Lecturer's Perspective. In Proceedings of the 2nd International Conference on Education and Multimedia Technology, New York, NY, USA, 2–4 July 2018; ICEMT '18; pp. 1–4. [CrossRef]
38. Tay, J.; Goh, Y.M.; Safiena, S.; Bound, H. Designing digital game-based learning for professional upskilling: A systematic literature review. *Comput. Educ.* **2022**, *184*, 104518. [CrossRef]
39. Díaz, J.; López, J.A.; Sepúlveda, S.; Ramírez Villegas, G.M.; Ahumada, D.; Moreira, F. Evaluating Aspects of Usability in Video Game-Based Programming Learning Platforms. *Procedia Comput. Sci.* **2021**, *181*, 247–254. [CrossRef]
40. Lindberg, R.S.N.; Laine, T.H.; Haaranen, L. Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games. *Br. J. Educ. Technol.* **2019**, *50*, 1979–1995. [CrossRef]
41. Cavalcanti, A.P.; Barbosa, A.; Carvalho, R.; Freitas, F.; Tsai, Y.S.; Gašević, D.; Mello, R.F. Automatic feedback in online learning environments: A systematic literature review. *Comput. Educ. Artif. Intell.* **2021**, *2*, 100027. [CrossRef]
42. Mingins, C.; Miller, J.; Dick, M.; Postema, M. How We Teach Software Engineering. *JOOP* **1999**, *11*, 64–66.
43. Nuci, K.P.; Tahir, R.; Wang, A.I.; Imran, A.S. Game-Based Digital Quiz as a Tool for Improving Students' Engagement and Learning in Online Lectures. *IEEE Access* **2021**, *9*, 91220–91234. [CrossRef]
44. Shaw, M. Software Engineering Education: A Roadmap. In Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, 8–13 November 2000; ICSE '00; pp. 371–380. [CrossRef]
45. Nosál', M.; Sulír, M.; Juhár, J. Source code annotations as formal languages. In Proceedings of the 2015 Federated Conference on Computer Science and Information Systems (FedCSIS), Lodz, Poland, 13–16 September 2015; pp. 953–964. [CrossRef]
46. Diaz, C. Using static site generators for scholarly publications and open educational resources. *Code4Lib J.* **2018**.
47. Leinonen, J.; Denny, P.; Whalley, J. A Comparison of Immediate and Scheduled Feedback in Introductory Programming Projects. In Proceedings of the 53rd ACM Technical Symposium on Computer Science Education—Volume 1, New York, NY, USA, 2–5 March 2022; SIGCSE 2022; pp. 885–891. [CrossRef]
48. Kian, T.W.; Sunar, M.S.; Su, G.E. The Analysis of Intrinsic Game Elements for Undergraduates Gamified Platform Based on Learner Type. *IEEE Access* **2022**, *10*, 120659–120679. [CrossRef]
49. Porubän, J.; Nosál', M. Generating Case Studies from Annotated Sources Codes. *J. Comput. Sci. Control Syst.* **2013**, *6*, 81–86.