education sciences

MDPI

Article

Toward an Innovative Educational Method to Train Students to Agile Approaches in Higher Education: The A.L.P.E.S.

Jannik Laval [1], Anthony Fleury [2], Abir B. Karami [3], Alexis Lebis [2], Guillaume Lozenguez [2,*], Rémy Pinot [2] and Mathieu Vermeulen [2]

[1] INSA Lyon, Université Lyon, Université Lumière Lyon 2, Université Claude Bernard Lyon 1, DISP, EA4570, 69676 Bron, France; jannik.laval@univ-lyon2.fr
[2] IMT Lille Douai, Institut Mines-Télécom, Université Lille, Centre for Digital Systems, F-59000 Lille, France; anthony.fleury@imt-lille-douai.fr (A.F.); alexis.lebis@imt-lille-douai.fr (A.L.); remy.pinot@imt-lille-douai.fr (R.P.); mathieu.vermeulen@imt-lille-douai.fr (M.V.)
[3] Smart and Sustainable Cities Team, Faculty of Management, Economics & Sciences, Lille Catholic University, F-59000 Lille, France; abir.karami@univ-catholille.fr
* Correspondence: guillaume.lozenguez@imt-lille-douai.fr

Abstract: Introduced in 2013, the A.L.P.E.S. approach (AgiLe aPproaches in higher Education Studies) aims to apply agile practices to teaching. Agile approaches are project management practices for IT development. More pragmatic than traditional methods, they allow to be closer to the applicant and to involve him/her as much as possible. They offer a great reactivity and a good adaptation to best meet the needs. They are used today in a large part of IT companies. Largely inspired by agile approaches, the A.L.P.E.S. approach allows the teaching of project management in a transverse way to a main course. It makes teaching more flexible and more adapted to the students. In this article, we describe the approach. We describe the tools, the process of creating a course, and the process of running a course.

Keywords: teaching methods; pedagogical innovation; creativity; pedagogy support

Citation: Laval, J.; Fleury, A.; Karami, A.B.; Lebis, A.; Lozenguez, G.; Pinot, R.; Vermeulen, M. Toward an Innovative Educational Method to Train Students to Agile Approaches in Higher Education: The A.L.P.E.S. *Educ. Sci.* **2021**, *11*, 267. https://doi.org/10.3390/educsci11060267

Academic Editor: Eleanor Dommett

Received: 3 May 2021
Accepted: 24 May 2021
Published: 28 May 2021

1. Introduction

In 2001, a group of leading IT developers designed and wrote the agile manifesto [1]. This manifesto aimed to propose a paradigm for software development around four founding principles as an alternative to widely used classical IT project management methods (Such as the v-Model for Software Development [2]). Today, agile approaches and methods (with, as an example, the well-known *SCRUM* [3]) based on this manifesto are widespread in the world of IT development but also in other sectors of the enterprise. Therefore, it would seem judicious to initiate future engineers (as project contributors for IT but also in other domains) to agile approaches. The idea we defend is not only teach agile project management, but to integrate it into a pedagogical approach.

As project-based pedagogy is widely used in higher education and has many intrinsic qualities, its adoption for the design of such a modality seems interesting. In fact, the question of a project-based pedagogy integrating the concepts of agile approaches is acutely raised: (i) how could agile approaches be integrated into project-based learning? (ii) could agile approaches be taught through project-based pedagogy?

In this article, we aim to focus on two hypotheses: (1) a course in higher education can be transformed to a project-based course that introduces agile concepts to students in addition to main concepts; (2) a well-defined model and the associated method can assist teachers in redesigning their courses to introduce agile concepts.

To this end, a model and the associated method to design agile project-based courses have been proposed to the French community [4]. This approach has been tested, among other schools and universities, at Mines Douai, an engineering school, since September

2014 in a course on database management (computer science), and at the University of Lyon 2. The results obtained validated its value both at the student and teacher levels [5].

On the strength of this positive feedback, we then informed and trained volunteer teachers and applicants to the approach. Several of them adopted and adapted the A.L.P.E.S. in their teaching. Building on this expertise, this paper proposes a presentation of A.L.P.E.S. through classical agile notions and tools (mostly from *SCRUM*) integrated in a teaching purpose.

The paper is organized as follows. After this introduction, we describe the context of the study in Section 2. In Section 3, we describe the tool philosophy used to create an A.L.P.E.S. course. Section 4 presents the processes to build and to follow a course. Section 5 gives some suggestions and advice on how to apply the method. A case study of the creation of a course from scratch is detailed in Section 6. We finish the paper with the conclusion in Section 7.

2. Context

To develop an approach that teachers can understand and use, it was necessary to review experiences in Project-Based Learning (PBL) [6–8] and agile project management [3,9]. On the one hand, the integration of agile approaches in education could have many advantages comparatively to the advantages of software development. On the other hand, PBL is widely used in many forms in higher education and would take benefice from the integration of agile notions.

2.1. Project-Based Learning

Project-Based Learning (*PBL*) is a model that organizes learning around projects. Projects in education is defined as complex tasks, based on challenging problems, that involve students in design, problem-solving, decision-making, or investigative activities [6]. In such a project, a student works alone or they form a project group autonomously with regard to the teacher during a determined period of time. A project ends with the delivery of a realistic product, document, report or presentation.

In the last two decades, *PBL* continued to gain in success and popularity in higher education. This method is well accepted by teachers, mainly because most of the projects are based on authentic content and situations. Thus, *PBL* aims to prepare students to realistic problems. However, as software development demonstrates it, there are many ways to manage projects.

2.2. Agile Project Management

Agile Manifesto [1] lays the foundation for a new paradigm for computer software or IT development. Agile approaches for project management were born from a need in computer science to continuously adapt projects to the client's and/or users' needs, emphasizing the human, and human interactions, at the centre of the project. Agile project management needs concepts and methods that assists the different participants (designers, developers, end users, etc.) to communicate during the phases of the project and to question the next developments to perform.

To develop the agile principles, methods are designed like *SCRUM* [3] to assist developers and project managers to create an agile experience. *SCRUM* provides a framework and tools to monitor the progress of the project.

A key feature in agile project management is to permit the teams to develop their project in an iterative way. The project timeline would be interrupted by several releases, where the product is fully operational on its developed components. The frequency of releases would be different from an approach to another. However a common central difficulty lies in the definition of the project components to develop (User Stories in *SCRUM* terminology) in a way to make an iterative process possible.

Tools issued from agile approaches like *SCRUM* are very interesting and may be used in a large variety of situations. Mainly, some tools, like planning board, help to organize and

follow the project. They contribute to the understanding of concepts associated with agile methods. Several of those tools will be selected and included in the pedagogical method.

2.3. Agile Approaches in Higher Education

SCRUM was the basis of some experiments like Lego for *SCRUM* [10,11] or courses to teaching software engineering [12]. In these approaches, the goal is to directly teaching agile project management: it's the main pedagogical objective of these courses. It seems difficult to adapt these examples to disciplines other than computer science or IT.

A.L.P.E.S. [4] is an adaptation of these approaches and their associated tools for a pedagogical approach dedicated to higher education, and this, for different thematic or disciplinary. This approach is based on the principle of inverted classes and introduces online documents that can be progressively consulted [13]. Therefore, A.L.P.E.S. lies in a socio-constructivist approach [14]; and more specifically in the project-based pedagogy paradigm. Taking benefits from both this paradigm and the ground principles of peer programming [15], A.L.P.E.S. defines the work sessions in the form of practical work and emphasizes collaborative work in groups (often in pairs).

However, defining a course in agile-project-based learning approach remains a challenging process for the teachers. As 'real professional project', it requires teachers to master the project-component notion to allow an iterative process (for students' project and competences in parallel). The remains of this paper focus on providing the basis for the notions of iterative development and User-Stories (the *SCRUM* project components) through an appropriation of agile tools and process in teaching purpose and a presentation, of courses' creation with examples.

3. A.L.P.E.S. through Agile Notions and Tools

Our method is based on PBL and integrate tools with the respect of the agile manifesto. It could be used with different disciplinary and allows teaching agile approaches in many contexts. We capitalize on more than six years of use of A.L.P.E.S. in different courses and thus, we propose a method to help teachers in designing their courses with agile approaches.

Most of the tools presented here are derived from agile projects and more specifically from *SCRUM* Method [16]. Several software solutions and web applications are available permitting to put them in practice. However, this section is more about the description of the tools philosophy than its implementation throw devices, applications or web services.

3.1. Time Decomposition

Agile Software Development's main assets come from iterative decomposition of the software development process. As a result, the first set of tools consists in defining rules and advises for deadlines and for specific sessions dedicated to development, project management and team organization.

Sprint: In *SCRUM* terminologies, *Sprint* identifies an iteration in Agile Software Development. *Sprints* are processed between two software releases planned at regular deadlines. It is mainly composed of development sessions between a *Sprint Planning* session and a *Sprint Review* session. The *Sprint Planning* consists in defining which component (*User Stories*) would be developed during the *Sprint*. The *Sprint Review* consists in presenting developed elements ideally with functional demonstrations. These both sessions that delimit a *Sprint* include the project stakeholder, generally the teacher in a pedagogical purpose. A sprint, in A.L.P.E.S., is considered as a course session. It takes generally 2 or 4 h.

Stand-up Meeting: It represents very short sessions permitting the team to state the progress, raise problems and distribute the tasks between two development sessions. Agile Software Development advises generally that *Stand-up Meetings* take place the first 15 min of a day's work. It could be referred to as *daily meeting*. This exer-

cise would be used especially in projects involving a 'large' student team (4 students and more) to learn how to coordinate the effort.

Time boxes: It represents the lower level of time decomposition, it aims to regulate development sessions. The main idea is that developers or students cannot be concentrated on their tasks during long sessions without interruption. As unexpected interruptions (e.g., mails, smartphones, and other social and professional network alerts) become more and more common and invasive and call for such time decomposition. Time breaks are scheduled all over a practical session to optimize the efficiency of work time. A classical model is based on 5 min time break every 25 min of work (Pomodoro method [17]). Those small-time breaks are used to do anything not related to the development of the current task while a 100% concentration is expected from the students in the other 25 min. Then, larger time breaks of 15 min are scheduled every two hours in case of long teaching courses.

3.2. Project Monitoring

Project monitoring tools permit the team to visualize the direction/goals of the project, the current project development state and their efficiency. Those tools could simply take the form of documents in the project directory.

Project-Book: This tool lists all the elements expected to compose the project realization. Those elements would be enunciated as *User Stories* (the next subsection goes into this notion in depth). *Project-Book* has an equivalent in *SCRUM*, the *backlog product* to state the advancement of the project.

In the end, it should look like the project scope statement. The main difference is that *Backlog product* in *SCRUM* or the *Project-Book* in teaching scenarios are not required to be completed at the beginning of the project/course. The document could evolve during the project development while it provides a vision of the project direction and goals and a clear definition of the elements to handle, at least for the next release.

From a teaching point of view, course and practical sessions involved in the project do not have to be completely defined at the beginning. Indeed, *SCRUM* recommends that the *User Stories* of the project should be organized according to the project's functionalities, and ordered by priorities. Thus, functionalities with high priority would be defined with detailed *User Stories* and developed first. Other functionalities would be only briefly presented and would match for instance optional notions that students could acquire during the course. Those functionalities would be detailed with a list of *User Stories* to develop only if/when students reach a certain point in their project development. Then, the evolution could be initiated from the teachers or from the students, if it is expected that the students take an active role in the project management.

Planning Boards: Boards are core tools for teams of students to organize their work and to communicate with teachers. The planning board (Figure 1) is decomposed in columns representing each *Sprint*, generally a session for teaching purpose.

Each column contains the *User-stories*, represented as a Post-it, dedicated to the *Sprint*. At a certain time step of the project development, this board permits a clear overview of what is already done (elements in past sprint) what is actually processed (elements in the current sprint) and what is planned to be performed later (elements in future sprint). It allows one to follow the overall progress and gives students visibility on the course objectives.

The *planning board* is questioned and updated only during the *Sprint Planning* session with the possibility to modify the columns mapping current and future *Sprints*. The *Sprint Planning* contracts on the *planning board* what the student team will perform during the next *Sprints* (generally at the beginning of a course session and for the remainder of the course session).

Task Boards: The task board (Figure 2), one per student team, containing 3 columns: *TO DO*, *DOING* and *DONE*. Each column includes the elements (*User-Stories* or associated tasks) relatively to their status: not yet investigated, currently processed or terminated.

DOING column is also composed with a *HELP* area which is particularly useful for interaction with the teachers. The main reason the students put items in waiting status is that they require a teacher to help or to validate.

During a *Sprint*, Post-its modelling *User-Stories* and tasks will navigate from one area to another. The task board allows one to visualize the status of the students in a session.

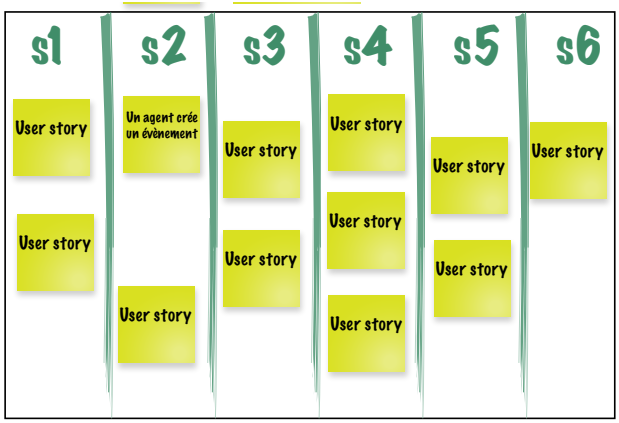

Figure 1. An A.L.P.E.S. Planningboard.

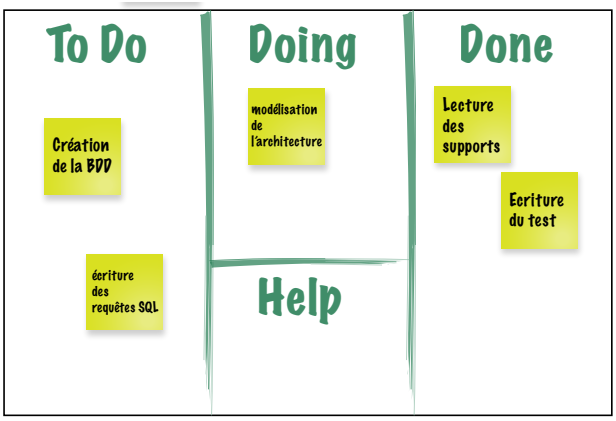

Figure 2. An A.L.P.E.S. Taskboard.

3.3. User-Story

The time decomposition and the project monitoring allows the students to incrementally validate their project development and their competences. In practice, that strongly depends on a 'good' definition of the *User-Stories*.

In *SCRUM* approach, a *User-Stories* represents a component of a software application. It is represented by a simple sentence that describes the expectation of the component to be developed. A *User-Story* illustrates a user's need. Each *User-Story* is noted on a Post-it and will be stored in the Product-Book.

To design a *User-Story*, the product owner (one of the project development roles, presented in the next subsection) should start from the "user need" and s/he should write a sentence with this structure: "As a *[role]*, I want to *[do task]*, so that *[a goal of the action]*". This way, the *User-Stories* is very close to a test case. The developer/student could play the specific user "role" and try to do the "task" in order to validate the good development of the required component(s). The "goal of the action" is also very important. It allows the

developer to have a full comprehension of the purpose of the developed component(s) for its own satisfaction and for a better interaction of the component(s) in the project.

In order to write correctly a *User-Story*, we recommend that it should be written as INVEST:

- Independent: each *User-Story* should be as independent as possible in order to be able to do one before the other.
- Negotiable: as long as it has not been started, it should be possible to modify a User Story.
- Valuable: The *User-Story* must bring value to the student.
- Estimable: a User Story must be estimable in terms of complexity. This is a rather complex point because students are learning. It is difficult for them to estimate the complexity of a task.
- Small: The smaller the User Story, the simpler and clearer it will be. The student gains confidence.
- Testable: For each *User-Story*, objective test criteria must be put in place to verify that the knowledge assimilated is correct.

Finally, the *User-Stories* can be completed in the project book with all the required information to develop the relative components in a more or less guided way.

In the context of a course, a *User-Story* will also represent the instantiation of pedagogical objective(s). This objective could clearly be established for the students on the "goal of the action". The *User-Story* division allows the teacher to follow the comprehension of the students as much as possible.

In fact, the *User-Story* is a central element of and A.L.P.E.S. approaches, because it allows a teacher to reengineer a course in terms of its actual application.

3.4. Roles

Roles permit differentiating actors and responsibilities in SCRUM project development. The kind of role an actor has is directly connected to the position of the person relatively to the project (developers, stakeholder, hierarchical supervisor, etc.). Classical project management is relaying on a unique project manager and this vision is common among the students who aim to become project managers. Attributing roles permits the students to apprehend the first distribution of the work and the responsibilities over the team.

Developers: All the members of the group working on developing the project. Each of them can share a second role in the following.

Scrum Master: He is responsible for the good application of the rules governing the developers. Technically, he presides the different sessions of the development process (*Stand-up Meeting*, *Sprint Planning*, *Sprint Review*), he maintains the different charts and other tools permitting self-evaluation and self-organization of the group.

Product Owner: He would be the main interface with the outside world and typically the stakeholder in software development. If he is not a necessary part of the developers, he would be available to clarify any point in the *User-Story* to develop. Generally, the Product Owner maintains the *Backlog Product*.

Outsiders: This category regroups the actors outside of the group of developers, mainly composed of helpers. It includes naturally the stakeholder, resource people (tester, evaluator, expert). *Evaluator* is naturally played at some point by teachers, but takes benefit to be attributed to students. Evaluators provide a critical review over the achieved work from a global point of view or over a specific aspect of the project. Their presence highlights the necessity to document development and to provide working demonstrations and it helps for the dissemination of good ideas.

3.5. Good Practices

Agile Software Development provides a favourable environment to experiment some practices. Many of those practices can be qualified as 'good practices' considering the speed of their propagation over the developers across the world. Beyond a presentation of those practices to the students in order to prepare them for their future career, those practices bring added value, in a pedagogical point of view.

Pair programming: "Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in. The two programmers switch roles frequently." [15]. It is interesting to present how two students working together on a unique computer could be beneficial to the project.

Test-driven development: The main ideas defining test-driven development is that tests need to be set up before starting any development. It allows the students to ask themselves the right questions regarding the piece of codes they are going to provide. The tests clarify in which context the development would be used and define precisely how the developed element would be called. Furthermore, by frequently performing tests (at least one test per User-Story) the students are capable of identifying problems before to accumulate too many mistakes in their project advancement.

Versioning: It consists of fixing project advancement steps. It allows the developers to safely backtrack at any saved project advancement step, possibly in parallel timelines (branches). As a first result, it permits the students to backtrack to a working version for a demonstration, at any time. Through this mechanism it is also interesting to visualize the evolution of demonstrations. As a second result, versioning helps the students working on shared resources. Through this tool, they learn to alternate development sessions performed in parallel and merge.

3.6. Tools in Agile Teaching

Agile software development was created over four values [1] starting with "individuals and interactions over processes and tools". In a learning case, individuals are mainly the students, and it is more about agile teaching of software development than agile software development, but the spirit remains intact. The aim of the tools is to help the students to increase their competencies both in course notions and in software development. For the teachers, it is more about providing a frame for the course definition and management.

Each teacher can adapt those tools to his/her own teaching skill and in fact, the same teacher can use them with different modalities depending on the courses, the students or the context. Those modalities could even evolve during the course.

Generally, the central question is about the level of autonomy the teachers want to grant the students. Who plays the different roles? Who decides the *User-Stories* to tackle during each sprint? Who defines the *User-Stories*?

The secondary question is about the concrete solution to provide for each tool. For instance, the first solution for boards, is to use real boards that require a certain logistics in a teaching environment. A second solution is to use digital solutions (files in a shared repository, online board services, etc.) that require computing manipulations before to visualize and modify it. Whatever the design of digital solutions, their usage would remain less intuitive and natural than pens and papers.

4. Presentation of the A.L.P.E.S. Processes

A.L.P.E.S. consists of two very distinct processes that we have modelled just below in flow diagrams. We have identified two processes: the first is the process describing the conduct of a course, i.e. a set of sessions with identified pedagogical objectives. The second process is the process of creating a course in A.L.P.E.S. format.

In this section, a focus is made on a teacher gathering all the responsibility. In fact, even if the teachers plan to empower the students on their project management, it is suggested to prepare the course as if not. This way, the teachers would have an idea of the overall coherence between a project, a possible set of *User-Stories*, a planning board and their usage in the course's session.

4.1. Course Session Process

A course session is organized in three parts: (i) the beginning of the session, where the teacher organizes the objectives and students organize their time, (ii) during the session is how to organize the current session, (iii) after the session, where the teacher prepares the next session based on the student feedback.

Figure 3 shows the process as it should be executed. Each task has a goal, inputs and outputs, and participants. In each task, there is the letter T (teacher) or S (Student) to show who is the participant to the task.

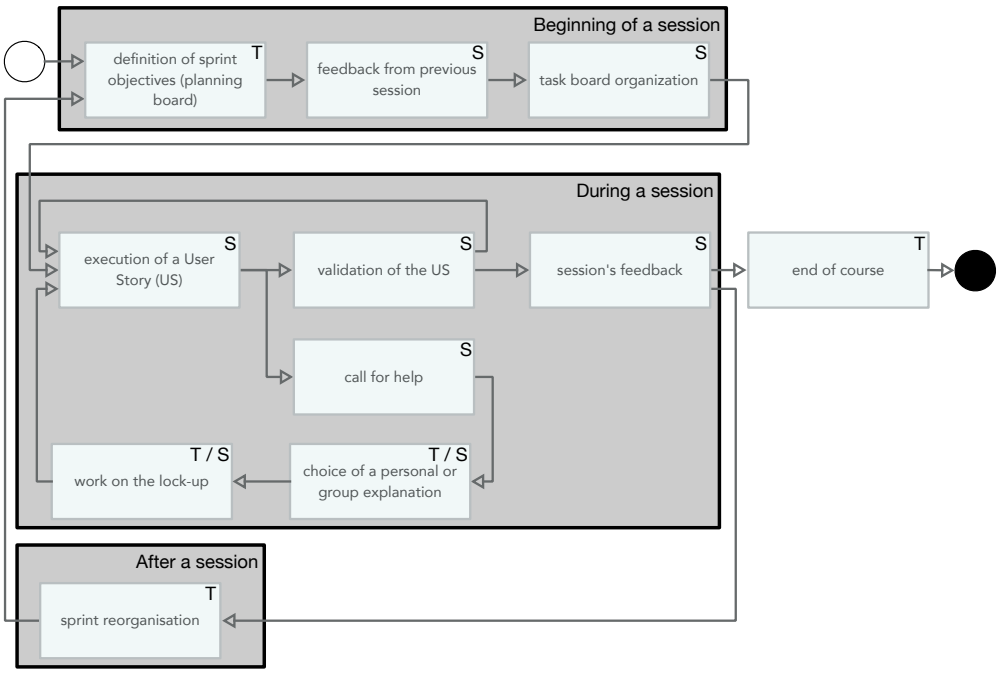

Figure 3. Process of an A.L.P.E.S. session (S for Student and T for Teacher).

4.1.1. Beginning of a Session

This group of tasks is the organization of the session. It should take 5 or 10 min.

- Definition of sprint objectives (planning board):
  - The objective of this task is to explain to the students what they will achieve during the session. This task is supported by the planning board, presented below.
  - As input to the task, we take the planning board from the course creation process (presented in the next section).
  - The output is a set of User Stories that the students will have to complete.
  - The actor is the professor. The students follow the instructions.
- Feedback from previous sessions:

. Once the students have a vision of the User Stories to be carried out, they will make the link with the knowledge and skills acquired in the previous sessions. This is the moment when the feedback from the previous session is given. We also analyse the charts (presented in Section 3.2). This feedback is interesting at this time because it allows connecting the sessions together and to re-mobilize previously acquired knowledge and skills.

. The input of this task is the set of User Stories to be performed during the session. From the point of view of tools, this represents a column of the planning board.

. The output is the same to-do list as the input.

. The actors are the students. Their activity during this phase is indispensable for the proper consolidation of knowledge. The teacher is on hand to answer questions.

- Task board organization:

. All the students select the User Stories to be done in the column corresponding to the session of the day, and place this list in the task board, in the “To do” column. In this list are also added to the User Stories that have not been completed during the previous session.

. The input of this task is the set of User Stories in a column of the planning board.

. The output is the task board ready to be used for the session.

. The actors are the students.

4.1.2. During a Session

- Execution of a User Story:

. The student chooses a task from the “To do” column of the task board, and places it in the “Doing” column. As the tasks are independent, he can choose the one he wants, knowing that all of them will have to be finalized. He’s directing the user story. When he has finished, he can validate the User Story (next task). If he has a problem, he asks for help by placing the User Story in the “Need help” column.

. The input is the task board in the following configuration: the “To do” column contains User Stories. The “Doing” column is empty. The "Done" column can contain tasks that have already been completed.

. The output is the Taskboard with a User Story placed either in the “Doing” column or in the “Need help” column.

. The actors are the students.

- Validation of the User Story:

. The student validates the User Story. In other words, depending on the subject of the course, the validation can be done either manually by the effective control of the teacher, or by an automated process via unit tests in computer science.

. The input is the User Story previously made.

. The output is the updated Taskboard: either the User Story is validated and goes in the “Done” column, or it is not validated and needs to be reworked.

. The actors are the students.

- Session’s feedback:

. When all the User Stories in the "To do" column are moved to the "Done" column, or when the end of the session has arrived, the students write the feedback of what they have learned from the session. The feedback is organized in the following way: "what I learned", "what I don't understand", "what I solved". In addition, students take the opportunity to update their task board and their updated planning board: The User stories in the "Done" column of the task board are moved to the current session column of the planning board. The user stories in the other columns switch to the next session column of the planning board.

  In parallel, the students update the charts related to the project (presented in Section 3.2).

. The input is the task board, up to date at the end of the session.

. The output is an empty task board, an updated planning board, updated charts.

. The actors are the students.

- Call for help:

  . When a student is stuck in a User Story to be made, they can ask the teacher for help through the *task board*. He/She can then start another User Story while waiting for the teacher's answers.

  . The input is a User Story on which the student has difficulties.

  . The output is an updated task board, with the User Story positioned in the appropriate location.

  . The actor is the student.

- Choice of a personal or group explanation :

  . When the teacher looks at the students' task boards, he/she will be able to identify the different User Stories present in the "Need help" area of the task board. He will be able to choose to give the necessary explanation either directly to the student or to the whole group by triggering a "stand-up meeting" (see explanation in Section 3).

  . The input is a blocking User Story.

  . The output is the updated Task Board.

  . The actors are the teacher and the students.

- Work on the lock-up :

  . The work on the Blocking User Story integrates the stand-up meeting, presented in Section 3 or an exchange between the teacher and the students, and personal work on the part of the student to resolve any misunderstandings.

  . The input is the Task Board with the blocking task in the "Doing" column.

  . The output is the Task Board with the blocking task in the "To Do", "Doing", or "Done" column or in the "Need help" area.

  . The actors are the teacher and the students.

4.1.3. After a Session

- It concerns the teacher's adaptation of the different User Stories. He must then move User Stories on the planning board, either moving them forward or backward according to the progress of the students. It can also add User Stories that seem necessary to improve pedagogy and the delivery of skills.

- The input is the session feedback. It is therefore a qualitative evaluation via this feedback and a quantitative evaluation via the analysis of the task boards that the teacher must carry out.
- The output is an updated planning board. It will be used for the next session.
- The actor is the teacher.

4.1.4. End of the Course

- As for the "Sprint reorganization" task previously presented, it concerns the adaptation by the teacher of the different User Stories according to a qualitative and quantitative evaluation. It leads to update each column of the planning board for the next class. Thus, from year to year, the teacher capitalizes and can refine his courses by being as close as possible to the needs of the students.
- The input is the feedback from the session and the modified planning board throughout the course.
- The output is an updated schedule board.
- The actor is the teacher.

4.2. Course Creation Process

Before a course session, the teacher has to create his course. This process is composed of five tasks that we define in this section. Figure 4 shows the process as it should be executed. Each task has a goal, inputs and outputs, and a participant. In this process, only the teacher (T) is involved.

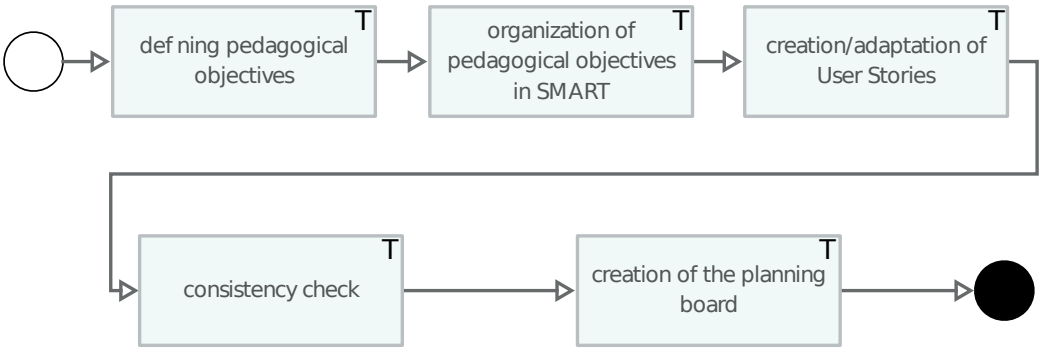

Figure 4. Process of an A.L.P.E.S. session creation.

4.2.1. Defining Pedagogical Objectives

- In this task, the teacher defines the different pedagogical objectives of the course. It must have 1 to 2 pedagogical objectives per session.
- The input is the name of a course.
- The output is a list of pedagogical objectives.

4.2.2. Organization of Pedagogical Objectives in S.M.A.R.T.

- In this task, the teacher describes his/her pedagogical objectives in a S.M.A.R.T. way:
  - Simple: the student must have the means to achieve it. Simplicity is synonymous with efficiency. Complexity slows or even blurs learning.
  - Measurable: an objective can only exist if it is measurable. This characteristic will then make it possible to evaluate the result in terms of outcome, and also in terms of knowledge acquisition.

. Ambitious: in order to get the student involved, the objective to be achieved must require a learning effort. It must also be accepted, i.e. the student must understand why he or she must achieve the goal.

. Realistic: If the goal is perceived by the student as impossible to achieve, he or she may become discouraged.

. Time-bound: The objective must be achieved within a reasonable period of time. That is to say that during a session, students should be able to achieve 1 to 2 pedagogical objectives.

- The input is a list of learning objectives.
- The output is a list of S.M.A.R.T. pedagogical objectives.

4.2.3. Creation/Adaptation of the User Stories

- Once the pedagogical objectives have been defined, the teacher can break them down into User Stories (presented in Section 3) while keeping the S.M.A.R.T. objectives. The S.M.A.R.T. characteristics are the same as before, except for the time-bound, which is defined as follows:

  Time-bound: The objective must be achieved within a reasonable period of time. The 25-min time-box is a good way to keep up with the students' pace and have a simple progress indicator.
- The input is a list of S.M.A.R.T. pedagogical objectives.
- The output is the Project-Book (presented in Section 3.1) listing the User Stories.

4.2.4. Consistency Check

- After constructing the list of User Stories, the teacher verifies that all user stories are consistent and independent.
- The input is a list of the User Stories.
- The output is an updated list of the User Stories.

4.2.5. Creation of the Planning Board

- When all User Stories are created, the teacher can create the planning board (presented in Section 3). He places the Users Stories in an order that allows him to advance pedagogically while keeping the students motivated.
- The input is a list of the User Stories.
- The output is the planning board.

5. Case Study: Creating a Course with A.L.P.E.S. from Scratch

Starting with the courses of Database Management System, we saw that the method was very efficient to help the students working on a computer science project [4,5]. In March 2014, when came the idea to create a course of Swift Programming Language, it was logical to initiate it using A.L.P.E.S. method. This section presents some feedback on this setup of a new course. We also regroup in this section feedback from another case study that concerns transforming an existing course to A.L.P.E.S.

5.1. Choice of the Subject, Creation of the User Stories

Compared to the adaptation of an existing course, going from a blank page to create the course is the simplest version of the A.L.P.E.S. use. Having nothing, it is simple to find a problem to solve that is interesting for the students (depend on the curriculum or on the population), that could be solved incrementally, and that will allow the teacher to show them the different aspects of the course (to acquire the targeted knowledge). Students have

previous knowledge related to the course; here they extend their knowledge on OOP by using a graphical interface and tactile devices.

One simple choice was to implement a game. The chosen game is 2048, very simple game to implement. The used development environment offers REPL (Read Eval Print Loop), a graphical tool able to compile online some code and even display some interfaces. In the first User Story, they use REPL to learn the language, its differences with Java that they learn before, and all the use of Xcode, Apple's development environment. The second User Story is devoted to learning to create graphical interfaces with which you can interact with gestures. In the third one, they start the creation of the cell class, the minimal entity of the game, and then the fourth is on the creation of the game board. The implementation of the rules is the fifth. Then, to motivate them, the last User Story is devoted to the improvement of the game. Following the course, they will create a first version of the game, that feels like one anyone can find on the web. What if now they add some features? Saving the board between execution, allowing the person to cancel a movement, changing the size of the board, creating some styles and decoration, all those things that they can ask the teacher how to implement. Each year, some other documentation is created to answer problems they could have.

5.2. Programming Knowledge

The first part of the project is the acquisition of minimal knowledge to be able to do something in the course: using the development environment and learn the programming language. This part is done fully online, using the flipped class model. They learn and they test at the same time and interact with the teacher when there are some points that are not clear. When a point is problematic for everyone, it is shared with all the class.

5.3. The Project Itself, Interaction with the Students

During the creation of their game project, they will use their board to represent their progression in the US, and they will, as in the first part, ask questions when there are. Globally, over the years, we have very interesting projects, ideas and the slides can be completed with new ideas that come to them. The projects that are evaluated are very good and the implication of the students is real.

We have to note that this course has been created in February 2015, the first-year of the language, with tools that were bugged. The use of the A.L.P.E.S. method made the course easier to handle with the different questions of the student raised by the bugs.

5.4. Students Feedback–Comparison with the Second-Year Course

After this course, in the following year, there is a second course of mobile development for iOS that is done by the same teacher. In this second course, another language is learnt, but the same interface is used. For those who follow the two courses, they largely prefer the A.L.P.E.S. one (the second course is more classical with a teaching part, exercises and labs). They have better comprehension of the language, they can do things quicker. And this feeling and feedback from the student are present since the beginning of the course.

5.5. Covid-19 Course Execution Style: Going with A.L.P.E.S. Methods Virtual

In March 2020, Covid-19 troubled the teaching all around the world. For France, universities were closed, and all the teaching went in distance learning mode. It was a very good experience for our knowledge of A.L.P.E.S. What happens for such course when it goes online? Is it simpler or harder to handle than a classic course? Equipped with Zoom and giving them a virtual machine to work, this experience was amazingly simple. All the A.L.P.E.S. implementation of this course makes it "simple" to follow with the teacher out. Creating on-the-fly rooms to answer questions individually as when we come to the student were sufficient. The project, in this setup, could not be on iOS as it needs a Mac to execute, it was a command-line games type on Linux. In the final, I had lots amazing projects, showing that they understood and also that they worked hard.

6. Feedback from Courses Transformation Experiences

We regroup in this section feedback from teachers in the process of transforming their courses into agile-project-based courses using A.L.P.E.S.: one course for the second-year undergraduate and another for the third-year undergraduate.

6.1. About Adapting and Personalizing the Process

A.L.P.E.S. is a rich approach with several concepts and tools. It is clear that teachers with experience in teaching with A.L.P.E.S. will have a higher level of mastery of its tools and concepts and better ability to adapt the process to their needs and to their student level.

For the third-year undergraduate, a course on Statistics and probabilities with R is proposed. The students were given the liberty to propose their own user stories following a list of conditions to respect. For the second-year undergraduate students, user stories are prepared by the teacher. A list of must do user stories are assigned for each session. To give some liberty for well-advanced students and more experience in autonomy for seeking information, a list of bonus user stories (ordered by difficulty) is provided to the class. Once they must do user stories are done, students can choose and add bonus user stories to their planning board.

For these courses, the incremental development cycle is respected with an iteration over each user story that includes the following steps (based on the spiral *SCRUM*):

- Seeking information: learn how to look for needed information to realize the user story.
- The conception or the modelling of the solution.
- Implementation of the solution.
- Testing and validation of the solution.

6.2. On the Choice of the Project Subject

The project should be considered as the spine for notion learning (hard and soft notions) for the entire course. Choosing the subject of this project must be given the necessary time. Here are some points to keep in mind during this process:

- Choose an attractive subject for all students or the majority of them.
- Choose an accessible subject: understanding the project subject and its purpose must not be an issue or reason for complexity.
- Easily divided in small accessible and comprehensible functionalities (user stories).
- The functionalities must be rich and numerous enough to cover all the notions (hard ones), potentially several user stories per notion.

The proposed project for the course of statistics is to analyse the logs of 1000 matches of a video game. Very positive feedback came back from the students on the course. The project subject had a very important positive effect on their motivation.

6.3. Flipped Classroom

Lots of benefits comes from the flipped classroom approaches [18]. The idea is for students to have the possibility to study the content on their own pace. The time shared with the teacher is for focusing on complex notions. Also, this will allow more time to exchange between students (and with the teacher) to open the horizon on the subject. The theoretical content can be transmitted in full to the students prior to the beginning of the first session. In this case, it can be considered as one of the resources for seeking needed information when achieving user stories all along the courses' sessions. However, in courses for undergraduate students, sending the theoretical content in full at once might be heavy on the students. The content, in this case, can be transmitted in smaller quantities before each session while focusing the transmitted content to the applied notions in the session in question.

6.4. Interaction and Feedback

- Interaction with students and collect their feedback: It is important to take into account this feedback to adapt the content of the course to better answer to the needs and capacities of the students. However, this should not cause incoherence in the evolution of the sessions. Some modifications are better to be noted for the next iteration of the course.
- Interaction with the other teachers, especially those responsible for courses where prerequisite notions are presented. Communication on what notions to concentrate on and what to take for granted is important information during the creation of the course, the user stories and the bonus user stories (that can be used as revision work for those in need).

7. Conclusions

In this article, we have presented the process of building a course in A.L.P.E.S. In previous articles [4,5], we discussed the implementation of the approach and the feedback from the students. This article consolidates the implementation of A.L.P.E.S.

In the short term, we would like to compare A.L.P.E.S. with other approaches, by carrying out studies on the most receptive subjects or student levels. Our long-term goal is to model the approach with an ontology.

The approach comes from the field, like all agile approaches. The next step is to model the principles in order to be able to share and exploit the project books. The objective is to share and instantiate courses according to the profile of the students, the pedagogical objectives, by questioning a knowledge base shared between the actors of the A.L.P.E.S. approach. The construction of an ontological knowledge base is a step which seems to us indispensable.

We built the A.L.P.E.S. approach by investigating the different principles and the different tools inherent to agile methods (cf. Section 3) through the investigation of pedagogical objectives.

We have classified them according to their intrinsic properties and the pedagogical objectives they give rise to. These classes, which we call "Principles" (cf. Figure 5), thus enable us to organize the structuring of the A.L.P.E.S. approach and condition the processes capable of being implemented for the development of A.L.P.E.S. courses.

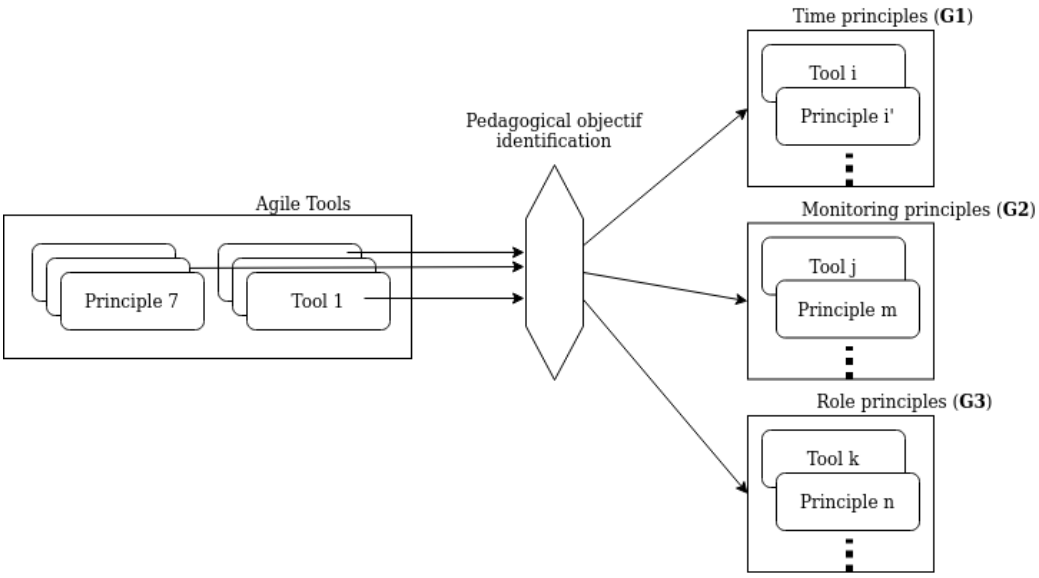

Figure 5. Course Definition Model.

In our work so far, we have been able to identify three main classes, namely (1) *Time principles*, (2) *Monitoring principles*, and (3) *Role Principle*, from the agile objects we have studied. Consequently, a course in the sense of A.L.P.E.S. is composed of a set of elements coming from the different classes of principles identified. Nevertheless, it is not excluded that as agile methods and practices evolve, new classes may appear, enriching the expressivity and possibilities of A.L.P.E.S.

This definition provides us with a framework for establishing the minimum conditions for belonging to the A.L.P.E.S. paradigm and, incidentally, that the approach we present is generally exploitable there.

We have not discussed here either the emerging specificities related to these processes, or their alterations, when a specific subset of principles is chosen to define a course. Rather, we have focused on identifying and defining the processes enabled by the full use of agile methods in a pedagogical framework.

Author Contributions: J.L. and M.V. created and designed the A.L.P.E.S. approach and wrote the previous articles. In this article, they describe the approach and define the processes. A.B.K., R.P., A.F. and G.L. applied the A.L.P.E.S. approach in their courses, had multiple feedback and adaptation of the method and tools during their different courses and were implicated in the improvement of the model based on these feedback. A.L. and G.L. defined the A.L.P.E.S. model. All authors have read and agreed to the published version of the manuscript.

Funding: This research and the Article Processing Charges was funded by APACHES, an I-SITE ULNE project, grant number FIPE18-007-VERMEULEN.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Beck, K.; Beedle, M.; Van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; et al. Manifesto for Agile Software Development. Technical Report, 2001. Available online: https://agilemanifesto.org/ (accessed on 27 May 2021).
2. Forsberg, K.; Mooz, H. The Relationship of System Engineering to the Project Cycle. *INCOSE Int. Symp.* **1991**, *1*, 57–65. [CrossRef]
3. Schwaber, K.; Beedle, M. *Agile Software Development with Scrum*; Prentice Hall: Upper Saddle River, NJ, USA, 2002; Volume 1.
4. Vermeulen, M.; Fleury, A.; Fronton, K.; Laval, J. LES ALPES: Approches agiles pour l'enseignement supérieur. In Proceedings of the Colloque Questions de Pédagogies dans l'Enseignement Supérieur (QPES 2015), Brest, France, 17–19 June 2015; pp. 243–248.
5. Vermeulen, M.; Laval, J.; Serpaggi, X.; Pinot, R. Soyez agiles dans les ALPES! Une pédagogie en mode agile. In Proceedings of the Colloque Questions de Pédagogie dans l'Enseignement Supérieur (QPES 2017), Grenoble, France, 13–16 June 2017.
6. John, W.; Thomas, W. *A Review of Research on Project-Based Learning*; The Autodesk Foundation: San Rafael, CA, USA, 2000.
7. Karabulut-Ilgu, A.; Jaramillo Cherrez, N.; Jahren, C.T. A systematic review of research on the flipped learning method in engineering education. *Br. J. Educ. Technol.* **2018**, *49*, 398–411. [CrossRef]
8. Kingston, S. *Project Based Learning & Student Achievement: What Does the Research Tell Us*? PBL Evidence Matters; Buck Institute for Education: Novato, CA, USA, 2018; Volume 1.
9. Highsmith, J.; Cockburn, A. Agile software development: The business of innovation. *Computer* **2001**, *34*, 120–127. [CrossRef]
10. Paasivaara, M.; Heikkilä, V.; Lassenius, C.; Toivola, T. Teaching students scrum using LEGO blocks. In Proceedings of the Companion 36th International Conference on Software Engineering, New York, NY, USA, May 2014; pp. 382–391.
11. Ouitre, F.; Lambert, J.L. Le lego4scrum, un dispositif agile pour enseigner le management de projet-Innovation Pédagogique. In Proceedings of the Colloque Questions de Pédagogie pour l'Enseignement Supérieur (QPES 2015), Brest, France, 21 June 2015.
12. Mahnič, V. Scrum in software engineering courses: An outline of the literature. *Glob. J. Eng. Educ.* **2015**, *17*, 77–83.
13. Bishop, J.L.; Verleger, M.A. The flipped classroom: A survey of the research. In Proceedings of the ASEE National Conference Proceedings, Atlanta, GA, USA, 22 June 2013; Volume 30, pp. 1–18.
14. Jonnaert, P. *Compétences et Socioconstructivisme: Un Cadre Théorique*; Groue De Boech: Bruxelles, Belgium, 2009.
15. McDowell, C.; Werner, L.; Bullock, H.; Fernald, J. The effects of pair-programming on performance in an introductory programming course. In Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, New York, NY, USA, February 2002; pp. 38–42.

16. Sutherland, J.; Schwaber, K. The Scrum Guide. In *The Definitive Guide to Scrum: The Rules of the Game*. Available online: https://scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf (accessed on 27 May 2021).
17. Cirillo, F. *The Pomodoro Technique*; Creative Commons: San Francisco, CA, USA, 2009.
18. Butt, A. Student views on the use of a flipped classroom approach: Evidence from Australia. *Bus. Educ. Accredit.* **2014**, *6*, 33–43.