ACTIVE LEARNING AND INTRINSIC MOTIVATION IN DIFFERENT DOMAINS: APPLYING A CONSISTENT SET OF PRINCIPLES OVER DISTINCT SETS OF LEARNING GOALS

Marcelo Hashimoto
Insper (BRAZIL)

Abstract
In this paper, we describe a general framework for designing courses based on active learning and intrinsic motivation and, to illustrate the applicability of the framework over different sets of learning goals, we also describe three different courses designed from it. We start by presenting the motivation for instructional design based on active learning and intrinsic motivation and establish the specific focus adopted. Afterwards, we describe the specific details of each course. To conclude, we summarize the preliminary findings obtained after ministering these courses and propose future steps.

Keywords: active learning, intrinsic motivation, engineering, project-based learning, flipped classroom.

1 INTRODUCTION

Active learning is a broad term that characterizes teaching methods in which students actively participate in their learning process instead of passively listening [1]. In recent years, there has been an increase in course designs based on principles of active learning [2] due to several reported cases of correlation between the application of such principles and measurable improvements of student performance [1, 3, 4].

However, actively participating in the learning process depends on being motivated to do it in the first place, preferably without the need of external rewards or punishments. This is why several initiatives in education innovation pay special attention to intrinsic motivation [5], the drive to engage in a behaviour that rises from within the individual. There has been successful cases of measurable increase in student performance through the development of a learning environment that stimulates the three aspects of intrinsic motivation: autonomy, competence, and relatedness [6].

In order to adequately apply active learning and intrinsic motivation in the classroom, a professor first needs to understand that flexibility is a key aspect of effective instructional design [7, 8]. Neither active learning nor intrinsic motivation should be interpreted as a one-size-fits-all toolbox of methods, but as a set of principles that only limit the number of possible methods. The specific choice for each activity of each course depends on the idiosyncrasies of the skills and contents involved in the learning goals and, more importantly, on the specific characteristics of the students.

In this work, we analyse and compare the application of a consistent set of principles, from the theories of active learning and intrinsic motivation, over three courses of the new Engineering program at the Insper institution in São Paulo, Brazil [9]. Despite being designed and ministered by the same professor, and to the same students, the three structures had several differences. However, we argue that they all follow successfully the original set of principles.

In Section 2, we provide an overview of the framework we used for course design. In Sections 3, 4, and 5, we describe in details each one of the three courses: Co-Design of Applications, Agile Collaborative Development, and Programming Challenges. Each section includes the goals, particularities, structure, activities, and preliminary results of its respective course. We conclude the paper with a summary of these preliminary results and promising possibilities of future actions.

2 GENERAL FRAMEWORK

In this section, we present the set of principles that guided the design of all three courses described in this paper. We describe the specific definition of active learning considered, and list the three aspects of intrinsic motivation that we consciously focus on.
2.1 Active learning

There is no single, universally accepted definition of active learning, but we choose a classic sentence from Bonwell and Eison as a base: "Instructional activities involving students in doing things and thinking about what they are doing" [10].

By “doing things” we do not mean any activity, but specifically activities that are somehow effective to achieve relevant learning goals. In order to ensure that this is always the case, we follow the process of backward design [8]: we start by identifying the desired results (learning goals), proceed to determining acceptable evidence (assessments), and only afterwards we plan the activities.

By “thinking about what they are doing” we mean that the learning goals should not be focused on lower cognitive levels. We use the Revised Bloom Taxonomy [11] to ensure we always define a set of learning goals that are concentrated at least on medium cognitive levels.

In summary, our focus on active learning can be defined as a conscious effort to propose activities that are particularly relevant to high cognition learning goals.

2.2 Intrinsic motivation

According to the self-determination theory in psychology [5], intrinsic motivation depends directly on three key aspects that can vary independently.

The first aspect is autonomy. A student feels more motivated when a higher autonomy is perceived with respect to the task. This autonomy can be present in several parts of the task: the goals, the technique or tools used, the time required to complete it, and the team working along are some examples of parts over which the student can have less or more autonomy. It is important to note that perceived autonomy does not necessarily mean real autonomy. It is common for professors to tell students that they can use any method to solve a certain problem, but the very nature of the problem severely restricts the universe of methods they can actually use.

The second aspect is competence. A student feels more motivated when the completion of a task represents a real possibility of obtaining new skills or improving existing ones. Such representation depends on a delicate middle ground between perceiving the task as too hard and perceiving the task as too easy. In the former case, the representation does not exist because it does not matter if the task provides or improves skills if it is impossible to be completed. In the latter case, the representation does not exist because completing a trivial task does not give the feeling of gaining something. Again, it is important to note that perceived difficulty is not necessarily the same as real difficulty.

The third aspect is relatedness. A student feels more motivated when a task can be associated with a concept that has personal value. A person with a strong sense of social justice, for example, can feel more motivated when the task to be completed can have a relevant impact on economic inequality. Relatedness is not necessarily related to the task itself: it can come, for example, from the desire of belonging to the group represented by the colleagues working on it.

According to the theory, if you successfully build a learning environment that stimulates these aspects in a student, you can make this student more intrinsically motivated to execute proposed activities.

3 CO-DESIGN OF APPLICATIONS

The Co-Design of Applications course occurs in the second semester of all three Engineering programs: Mechanics, Mechatronics, and Computer. It is a direct follow-up to Design Nature, a first semester course that introduces students to the traditional process of Engineering Design, with a strong emphasis on prototyping, and to basic principles of desirability. A key element of Design Nature is its final project: a bioinspired toy tested with children from an elementary school. While the students are coached enough to ensure positive feedback, they do not follow a complete process of understanding users, and therefore are likely to receive unexpected feedback as well. This experience is supposed to convince them of the importance of following such process, which is precisely what they are supposed to do in Co-Design of Applications.

3.1 Goals

The essential goals of the course are:
analyzing users from interviews and hypotheses;
− synthesize project questions from user analysis;
− create mobile application concepts from project questions;
− evaluate mobile digital prototypes from personas and scenarios.

The complementary goals of the course are:
− applying graphic design tools and web front-end technologies in digital prototyping.

In terms of content, the essential goals cover basic principles of Design Thinking, and the complementary goals cover basic principles of information architecture, Gestalt theory, CRAP principles [12], color theory, typography, usability, HTML, CSS, and JavaScript. In order to avoid cognitive overload, the course focuses on multidisciplinary breadth instead of specialized depth: relationships between different subjects are prioritized over details of specific subjects.

3.2 Particularities

Both Design Nature and Co-Design of Applications were inspired by Olin College courses [13]: the former is an adaptation of a homonymous course and the latter is an adaptation of a course called User-Oriented Collaborative Design. There are two key differences between Co-Design and UOCD that had impact on the course design. First, Co-Design occurs in the second semester while UOCD occurs in the fourth semester. Second, Co-Design targets a high-fidelity digital prototype of a mobile application while UOCD targets a low-fidelity prototype of any product or service.

The rationale for the first difference comes from the relatedness aspect of intrinsic motivation: by placing the course immediately after Design Nature, we use the experience of receiving unexpected feedback as a starting point, instead of letting its memories fade over time. Maximizing relatedness potential is particularly important because design courses are not traditionally associated to Engineering programs in Brazil, and therefore tend to suffer rejection from students.

The rationale for the second difference comes from the competence aspect of intrinsic motivation: on one hand, by requiring high-fidelity, we direct students to build tangible evidence of their accomplishments. On the other hand, by restricting to digital prototypes of mobile applications, we allow fast, interactive, and high-quality refinement of such evidence.

3.3 Assessments

The instruments for assessing the goals are:
− consistency of intermediate deliverables of projects;
− quality of final deliverable of projects;
− quality of short presentation about a project.

All instruments are applied over groups of students.

3.4 Structure

The students have a total of eight hours per week dedicated to the course. There are two mandatory meetings with the professor with two hours each, one optional meeting with one hour and a half, and two and a half hours available to work by themselves. The semester is divided in two modules.

The first module focuses on graphic design. It uses its own material, available to the students, and references to Codecademy courses [14] as additional support.

The second module focuses on Design Thinking. It uses its own material, available to the students, The Bootcamp Bootleg from d.school [15], and references to The Field Guide to Human-Centered Design from IDEO.org [16] and the Lean UX book [17] as additional support.

3.5 Activities

Both modules use Project-Based Learning [18], with different levels of autonomy.

In the first module, the project is a web portfolio for the work they have developed in the first semester of the program. The first meeting of the week is always dedicated to graphic design, and consists of a
short lecture followed by a long workshop to apply the concepts of the lecture in the project. The second meeting of the week is always dedicated to web front-end, and consists of a programming workshop to improve a mock portfolio made by the professor. Each meeting defines a deliverable representing an evolution of the project.

In the second module, the project is a prototype of a mobile application that satisfies a need of a user type. Each meeting is dedicated to a specific step of the Design Thinking process, and consists of a short lecture followed by a long workshop to apply the concepts of the lecture in the project. Each meeting defines a deliverable representing an evolution of the project. The students have almost complete autonomy over the choice of user type: the professor only requires that the students themselves do not belong to it. He also suggests types that are easy to find by foot in the neighbourhood.

In both modules, the students are organized in groups with no less than three and no more than four members. In the first, the groups need to have at least one student from each Engineering program, to encourage multidisciplinarity and break cliques. In the second, the students have complete autonomy over the choice of teammates, due to the complexity of the deliverables.

3.6 Preliminary results

According to qualitative feedback gathered at the end of the second semester of 2016, lack of relatedness was an issue: some students struggled to see the importance of the course in an Engineering program, while others struggled to see the importance of mobile applications for Mechanics or Mechatronics. They started feeling more comfortable when senior students described positive experiences associated with design in later courses or summer internships. Some also felt that the short lectures were unnecessary and that the programming workshops were too fast and lacked simple exercises.

Despite these issues, most of the projects had satisfactory quality and most of the students achieved the learning goals.

For the next iteration of the course, we plan to include presentations from senior students at the beginning of the semester, in an attempt to increase relatedness. We also plan to redesign the meetings to be more based on the Flipped Classroom model [19]: the short lectures can be replaced by online material, giving more space to small exercises.

4 AGILE COLLABORATIVE DEVELOPMENT

The Agile Collaborative Development course occurs in the third semester of the Computer Engineering program. It is a direct follow up to both Co-Design of Applications and Software Design, an introductory programming course that occurs in the first semester. In Co-Design of Applications, the students follow a process to integrate feedback from users into a project, but only build a prototype and do not consider feedback from other stakeholders. In Software Design, the students develop a functional software, but do not follow a software engineering process. In Agile Collaborative Development, they are supposed to fill these specific gaps.

4.1 Goals

The essential goals of the course are:

− follow principles of agile product development;
− use concepts of object-oriented programming;
− apply search algorithms over binary trees.

The complementary goals of the course are:

− explain basic concepts of software engineering;
− follow practices of agile software development.

In terms of content, the essential goals cover Lean Manufacturing [20], Scrum [21], class, object, attribute, method, inheritance, interface, polymorphism, exception, stack, queue, binary tree, depth-first search, and breadth-first search, and the complementary goals cover functional and non-functional requirements, business requirements and rules, version control and configuration.
management, waterfall model, unified process, pair programming, software testing, Test-Driven Development [22], Continuous Integration [23] and Extreme Programming [24].

4.2 Particularities

Agile Collaborative Development is very multidisciplinary: in traditional programs, its content is compartmentalized among four or five different courses. Although multidisciplinary approaches do not have enough space to reach the depth proposed by such programs, they address three common criticisms about compartmentalized approaches that the Insper professors have heard both in academic and professional environments.

The first criticism states that fundamental advantages of object-oriented programming, for example maintainability and adaptability, can only be noticed when developing an actual product in a software engineering framework.

The second criticism states that compartmentalization creates an artificial separation between architectural tasks and algorithmic tasks that does not exist in the real world.

The third criticism states that software engineering courses focus on obsolete theories instead of modern practices.

Addressing these criticisms can have impact on the relatedness aspect of intrinsic motivation, particularly for students who target a software development career.

4.3 Assessments

The instruments for assessing the goals are:

- quantitative performance on short quizzes;
- quantitative performance on long quizzes;
- quantitative performance on practical test;
- peer evaluation;
- consistency of intermediate deliverables of projects;
- quality of final deliverable of projects;
- quality of short presentation of a project.

The first four instruments are applied over individual students and the last three instruments are applied over groups of students.

4.4 Structure

The students have a total of eight hours per week dedicated to the course. There are two mandatory meetings with the professor with two hours each, one optional meeting with one hour and a half, and two and a half hours available to work by themselves. The semester is divided in four modules.

The first module introduces object-oriented programming, search algorithms, and software engineering. It uses its own material, available to the students.

The second module introduces agile product development and agile software development and consolidates object-oriented programming. It uses its own material, available to the students, and references to the Scrum book [21] as additional support.

The third module consolidates agile product development, agile software development, and search algorithms. It uses its own material, available to the students.

The fourth module consolidates agile product development and agile software development further, by introducing a real client into the process. It uses its own material, available to the students.

4.5 Activities

All modules have projects, but only the third and the fourth use Project-Based Learning. The first and the second use projects for consolidation instead of discovery.
The first module is strongly based on Flipped Classroom and loosely based on the eduScrum framework [25]: before each meeting, the students receive online material and are supposed to study it on their own. In the first fifteen minutes of the meeting, a graded but low-stakes quiz is used to assess their understanding of the material. In the remaining time, Coding Dojos [26] are used to implement a part of a project, with the professor acting as a Product Owner. The implementation is scaffolded to ensure consolidation of specific knowledge, but the professor keeps interference to a minimum and lets the dojos self-manage. Students are encouraged to use pair programming [27], but have complete autonomy over the choice of teammates.

The second module is based on active dynamics that have a history of success in professional training, for example the Lean Lego Game [28] and the Scrum Lego Game [29]. The students also implement a project on their own, with the professor again acting as a Product Owner. The implementation is less scaffolded, reflecting the progress they are supposed to have at this point.

In the third module, the students start with a practical test, to assess their understanding of the first two modules, and proceed to implement a project under the rules of the Scrum framework [21]. This project is their first contact with a mobile platform and they are supposed to figure it out on their own, with the professor acting as a coach.

In the fourth module, the students implement a project for a real client, obtained from institutional partnerships, with the professor again acting as a coach and institutional contact. They are not required to follow the rules of a specific framework, but they are supposed to follow agile principles.

Finally, because peer assessment is important for teamwork dynamics in Project-Based Learning [30], the students are asked to evaluate teammates after each of the last two projects.

4.6 Preliminary results

According to qualitative feedback gathered at the end of the first semester of 2017, many students reported a strong sense of competence and relatedness after the last two projects. The Flipped Classroom and Lego-based dynamics also received positive feedback. The main issue among some students was the sense of competence during the first module: they struggled to understand the online material and did not feel that the Coding Dojos were helping them to learn. Even students who had a good performance on quizzes and tests reported similar struggles.

For the next iteration of the course, we plan to redesign the first module in order to include more basic exercises as part of the Coding Dojos. We also plan to create feedback automation tools in order to provide quick response to students presenting difficulties at the beginning.

5 PROGRAMMING CHALLENGES

The Programming Challenges course occurs in the fifth semester of the Computer Engineering program. It can be seen as an indirect follow-up, as it provides a more detailed view of search algorithms that are introduced in Agile Software Development, but is mostly a standalone course due to its more theoretical focus.

5.1 Goals

The essential goals of the course are:

- summarize procedural algorithms as descriptive texts;
- simulate recursive procedural algorithms;
- analyze the correctness of procedural algorithms;
- analyze the efficiency of procedural algorithms;
- evaluate procedural algorithms in specific contexts;
- evaluate data structures in specific contexts.

The complementary goals of the course are:

- implement procedural algorithms with manual memory management;
- implement data structures with manual memory management.
In terms of content, the essential goals cover recursion, invariant, asymptotic analysis, linear and binary search, bubble sort, selection sort, insertion sort, merge sort, quick sort, stack, queue, binary tree, depth-first search, breadth-first search, linked list, binary heap, binary search tree, and hash table, and the complementary goals cover pointer, dynamic allocation, and dynamic programming.

5.2 Particularities

Co-Design of Applications and Agile Collaborative Development have a practical and multidisciplinary nature, allowing an easier focus on active learning and intrinsic motivation. In contrast, Programming Challenges has a theoretical and specialized nature, making such focus more difficult. Therefore, its design was inspired by Mathematics of Variation, a second semester course that also has a theoretical and specialized nature but was characterized by students as active and motivating.

A key element in Mathematics of Variation is the usage of printed handouts that students need to read and complete. These handouts are specifically designed to make the students discover important concepts by themselves, by experimenting with graphical tools and answering exploratory questions. Ideally, the professor just coaches the students through the handout without giving away the answers, but short lectures can be occasionally given to clarify difficult concepts. Programming Challenges adopts the same framework while adding the possibility of using web handouts with direct references to online tools and code repositories.

The course also gives special attention to the competence aspect of intrinsic motivation, by designing assessment tools that avoid the usual pressure of traditional tests. This strategy is another consequence of its theoretical and specialized nature: autonomy and relatedness are constrained by such nature, leaving competence as the aspect with better potential.

5.3 Assessments

The instruments for assessing the goals are:

- quantitative performance on checkpoint activities;
- quantitative performance on challenges.

All instruments are applied over individual students.

5.4 Structure

The students have a total of eight hours per week dedicated to the course. There are two mandatory meetings with the professor with two hours each, one optional meeting with one hour and a half, and two and a half hours available to work by themselves. The semester is divided in six modules but, unlike in Co-Design of Applications and Agile Software Development, the division is based on content. Each module uses its own material, available to the students, and ends with checkpoint activities.

The first module introduces the concepts of pointer and dynamic allocation. It is actually an intensive workshop shared with two other courses of the same semester, Hardware-Software Systems and Embedded Computing.

The second module introduces the concepts of recursion and invariant, using the algorithms bubble sort, selection sort, and insertion sort as examples.

The third module introduces the concept of asymptotic analysis.

The fourth module introduces the concept of divide-and-conquer, using linear search, binary search, merge sort, and quick sort as examples.

The fifth module introduces the concept of linear data structures, using stacks, queues, and linked lists as examples.

The sixth module introduces the concept of non-linear data structures, using binary heaps, binary search trees, and hash tables as examples.

5.5 Activities

A regular meeting closely follows the active framework described in Guide to Teaching Computer Science [31]: the professor briefly describes a problem or poses a question (trigger), proceeds to give the handouts and coach the students through it (activity), coordinates small debates to ensure that
important points are exposed (discussion), and ends emphasizing the main points (closure). When the implementation of the algorithm or data structure is particularly important, the next meeting is a programming workshop in which the students receive an incomplete code and need to fill the blanks in order to produce the correct answer.

The last meeting of a module is reserved for the **checkpoint**, a series of activities designed to assess the content of the module. A checkpoint is divided in three parts. The first part is an individual quiz with multiple choice questions, followed by its question-by-question resolution. If the student had difficulties with the quiz, the resolution might help with such difficulties and improve his knowledge for the next parts. The second part is a quiz inspired by peer instruction dynamics [32], in which the student has two opportunities to answer each question: the first attempt is completely individual and the second attempt is after discussing with the colleague at his side. Both the discussions and the resolutions are tools that can be used to improve knowledge for the last part, which consists of short dissertative questions. The general idea behind a checkpoint is gradually building the student confidence for next parts even if he did not necessarily had a good performance in previous parts. We use Plickers [33] for the second part, because their ludic nature can help in alleviating tension.

In parallel with regular activities, the students need to solve **challenges**, small programming problems based on famous programming contests [34]. The solution for these challenges are based on the classical algorithms and data structures taught in the course, but the difficulty is identifying exactly which one, or ones, should be used.

### 5.6 Preliminary results

According to qualitative feedback gathered at the end of the first semester of 2017, most students enjoyed the handout-based dynamics. The checkpoint activities also received positive feedback, with some students explicitly reporting that they did not feel like a test, despite being difficult. The main issue during the semester was adequately calibrating the difficulty and length of each handout, taking into consideration the differences among the students. In some meetings, the students with higher performance finished too early and did not have anything else to do until the end. In other meetings, the students with lower performance could not get past the first question of the handout, requiring a redesign of the next meeting to take their difficulties into more consideration.

For the next iteration of the course, we plan to redesign the handouts to make them more flexible. We will include both slower steps and advanced challenges in order to cover a large spectrum of students and be capable of adapting as needed during a meeting. We also plan to keep the checkpoint activities, as they were successful in increasing the sense of competence without hurting performance.

### 6 CONCLUSIONS

We described a general framework for designing courses, based on active learning and intrinsic motivation, and also described three courses that were designed through this framework. Because these courses have very different sets of learning goals, we could not use an one-size-fits-all toolbox of methods. Instead, we tried to use the best method for each part of each course, as long as it satisfied the same set of principles.

So far, the performance results and student feedback of these courses have been positive, at least from a qualitative and subjective point of view. Besides redesigning the courses based on such results and feedback, the next logical step is defining a framework for quantitative analysis of performance and motivation that allows us to objectively measure the impact of the framework.

### ACKNOWLEDGEMENTS

This work is still in the beginning and was only made possible with the help of both an institution and a team that believe in education innovation. The author would like to thank all the students, professors, and staff at Insper for all the valuable discussions, insights, and support.

### REFERENCES


