Hall of Shame & Fame: a pedagogical pattern for computer programming classes

Andrea S. Charão, Alberto F. Kummer Neto, Benhur de O. Stein, Patrícia Pitthan A. Barcelos
Federal University of Santa Maria

Pedagogical patterns propose to take advantage of the expertise in teaching and learning practices, in an organized manner that can be easily reused by educators. In this paper, we propose a pedagogical pattern targeted to teaching programming in higher level courses in Computer Science. This pattern is centered on the presentation and discussion of good and bad examples of code produced by the students, forming what was named respectively "Hall of Fame" and "Hall of Shame". Throughout the text, we present a characterization of HOFS pattern, relating it to other pedagogical patterns described in the literature. We also report its application in teaching object oriented programming in a higher education institution. The results indicate a positive evaluation by students and reveal the pattern helps to encourage good programming practices.

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computers and Information Science Education—*Computer science education*

General Terms: Languages, Education

Additional Key Words and Phrases: Educational Patterns, Language Patterns

ACM Reference Format:

Andrea S. Charão. 2016. Hall of Shame/Fame: a pedagogical pattern for computer programming classes – HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. (October 2016), 8 pages.

1. INTRODUCTION

Computer programming teaching and learning processes are a recurring subject of research [Pears et al. 2007; Robins 2010]. It is consensus that programming brings several difficulties to beginning students and even for those who have some previous experience, as there remain challenges as improving skills and learning new paradigms[Jenkins 2002; Robins et al. 2003; Lahtinen et al. 2005; Tan et al. 2009]. Faculty also face some challenges in this context, which includes decision making that impact on the entire process [Denning 1989]. Some frequent issues in this context are, for example: 'Which programming languages adopt to teach beginners?' and 'Which teaching and learning strategies are most effective?'

Looking for answers to this kind of questions, there are researchers that have dedicated to improve the understanding of cognitive processes involved in programming learning [Soloway and Spohrer 2013; Winslow 1996], while others investigate and propose ways to handle recurrent difficulties [Wilson and Shrock 2001; Caspersen and Kolling 2009]. To the latter, there are several authors that support an approach centered in "pedagogical patterns" [Sharp et al. 1996]. Pedagogical patterns were developed by a community interested in object oriented computer programming and gradually extended to a larger scope [Larson et al. 2008; Köppe 2015].

Author's addresses: {andrea, alberto, benhur, pitthan}@inf.ufsm.br, Federal University of Santa Maria, Rio Grande do Sul, Brazil.

This approach suggests the capture of good teaching and learning practices of specific domains in a organized and compact way that can be easily reused. Therefore, its main objective is to promote dissemination of good practices.

Since the initial propose, in 1996, several pedagogical patterns were described, cataloged and published, with several intentions. Specifically to programming learning domain, there are varied patterns, for example: *Mistake* [Bergin 2000], which proposes that the students develop programs (or other artifacts) with errors; and *Show Programming*, which suggests exposure of real time programming to the students–instead of slides about programming [Schmolitzky 2007]. According to authors of area [Bergin et al. 2012], some patterns may sound trivial to experienced educators, but they show up as useful references to less experienced ones, as well as a way to expand teaching skills. The succinct format used to present patterns ease its sharing; in other hand, the absence of complementary data about the evaluation of certain patterns can evoke questions about its effectiveness.

In this work, we present a pedagogical pattern to programming teaching in a higher course of Computation. This pattern–called HALL OF FAME & SHAME (HOFS)–is concerned on exposure and discussion of good and bad programming examples produced by the students. Its main objective is motivate the students to improve their programming skills. In addition to describe the pattern in succinct way, in a format that facilitates reuse, we also present a experimental evaluation of its application in a higher educational institution.

The text is organized as following. In the Section (1) we do a short historical review of pedagogical patterns for programming teaching, pointing the most relevant references about the subject nowadays; in Section (2) we present HoFS pattern in a common format used by community of the area; in Section (3) we describe the application and evaluation of the pattern on a higher educational institution; Lastly, we present our final considerations about this work in Section (4).

Pedagogical patterns [Sharp et al. 1996] emerged from discussions in object oriented teaching context and its technologies. The inspiration comes from so-called design patterns [Gamma et al. 1993], which consist of well established solutions for recurring problems in object oriented context. The initial idea of pedagogical patterns is, therefore, use a similar approach to catalog reusable solutions for common problems found on teaching of aforementioned paradigm.

Gradually, the idea of pedagogical pattern was extended to a larger scope, capturing solutions to problems found in different situations rather object orientation. Researchers and educators started to propose patterns in conferences, as PLoP (Pattern Languages of Programs) and EuroPLoP (European Conference on Pattern Language of Programs) and then, in more traditional conferences about computation teaching, such as SIGCSE (ACM Technical Symposium on Computing Science Education) and ITiCSE (ACM Conference on Innovation and Technology Science Education). The patterns were gathered in sites[1][2] and, more recently, several of them were presented in books [Bergin et al. 2012; Mor et al. 2014].

An important aspect of pedagogical patterns is its format of description, which is always succinct and organized to ease reuse. Although the format is not strict, it usually includes the following sections: pattern name, objective/motivation, applicability, structure, consequences, implementation, resources needed, examples and related patterns [Sharp et al. 1996]. Some author present the description in less sections, keeping more basic informations only [Schmolitzky 2007]. We choose to present the HOFS pattern within fewer sections to keep its presentation less fragmented. To ease communication, we use the pronoun “you” as a direct reference to teacher in Section (2).

2. THE HALL OF FAME & SHAME PATTERN (HOFS)

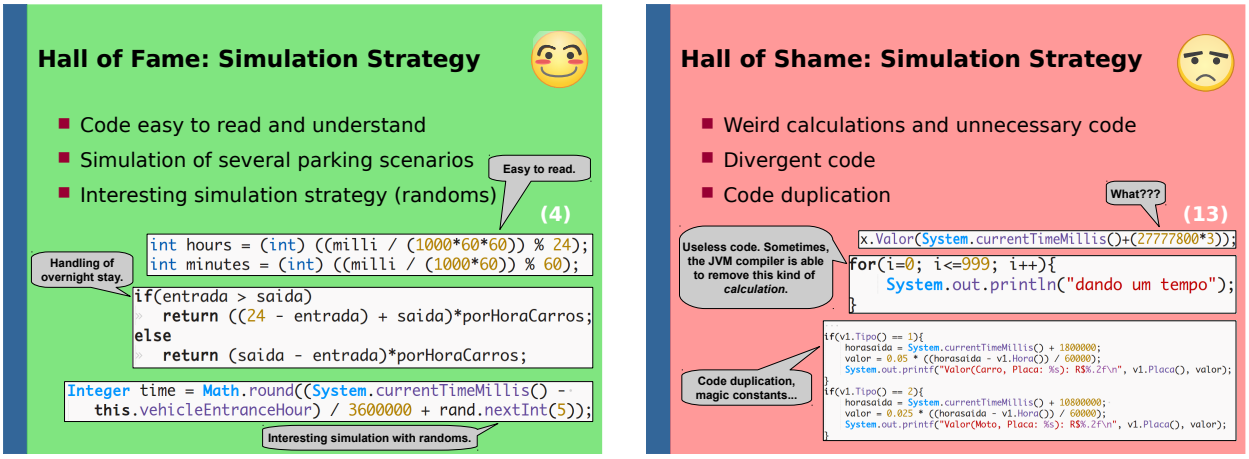
The proposed patterns is described as follows, using a similar format of presentation of [Bergin 2000].

- **Name:** HALL OF FAME & SHAME

[1] http://www.pedagogicalpatterns.org
[2] http://educationalpatterns.org

- **Problem:** The pattern is applicable to activities that involves design and decision making. In classroom, students might have difficulties to use or apply recently learned subjects to solve teacher assignments. In most of cases, they have not enough information to criticize their work-in-progress solutions, which can lead to adoption of 'trial-and-error' approaches. This is bad by itself, and such way of thinking mostly results in bad methodologies and habits. Such problems may happen again every time that a student face a new problem. The most common feedback procedures used by teachers are numbers (score), general comments, or standard solutions, which does not encourage students to think about how to improve their approaches of solutions development.
- **Context:** You are teaching a subject related to design or decision making, and your students have little skill with such topic. Suppose you are a teacher of programming class. For example, you may ask to your students to produce a software to solve a set of problems using a recently learned new programming technique, a new programming paradigm or a new programming language. Your objective is encourage the active thinking of students about how to solve such problems using the recently learned subject. You should be careful to craft the problem set to raise relevant questions about the study topic. Your classroom is not too big (it has 20 to 40 individuals). You want to encourage them to learn and adopt good problem solving practices, using you own expertise.
- **Solution:** After the students delivered their solutions to their assignments, do an analysis looking for good and bad solution practices. Desirably, allow students to deliver partial solutions before deadline set out to make them more active and involved with the activity [Larson et al. 2008; Köppe et al. 2015]. After each partial delivery, select some fragments of students solutions to compile an exposition separated in two "galleries", as shown in Figure 1: *Hall of Fame* (good practices) and *Hall of Shame* (bad practices). In classroom, show and comment about the selected solution snippets, asking for students to think and categorize the snippets as "Fame" or "Shame". You may use contrasting visual resources to identify both galleries. Note that, for the same problem, it is possible to find both good and bad solution approaches, which proposes that the gallery is not about good and bad programmers, but is about good and bad coding practices. You should also make the snippets anonymous. Whenever is possible, repeat this pattern. This approach stimulates the critical thinking of students, which is common habit between experienced programmers. The pattern provides feedback based in problems that the students dedicated themselves to solve. Instead of generic and abstract recommendations about problem solving methodologies, the pattern gives real examples from the own group of students. With a clear and reinforced distinction between Fame and Shame, it is expected a stimulus of cognitive processes, resulting in adoption of good practices in following assignments. It is not a objective of this pattern act as replacement to individual evaluation of programs (even in situations that it is not possible), but it is an option to promote quick and useful feedback to the group.
- **Side effects:** The pattern has its own limitations: (i) it demands a considerably amount of time to compile, expose and comment the galleries after each partial delivery, (ii) a homogeneity of experience of group tends to reduce diversity of solutions and (iii) the teacher should take care when referencing to snippets of solutions to not "attack" the students: making solution snippets anonymous could not be not enough since the author of snippet could recognize their own work. In this case, several occurrences of their solution in "shame" galleries can make the student lost his/her interest to follow up with the assignment.
- **Resource requirements:** It is recommended to expose the galleries with a multimedia projector, but any other multimedia resource could be employed to improve the discussion. Teacher may publish the material (in the course website, by e-mail, etc) for use outside of classroom. The problems proposed in the assignment should be specified to promote the variety of solutions. It could be useful some assistance from a monitor to prepare galleries and perform its exhibition–ideally from few days to up to a week after the delivery. It is a good idea to keep a repository of solution snippets of previous applications of pattern to enrich the discussions when students solutions are too much similar, for example.

- **Examples:** An example of application of HOFS is in teaching of procedural language C, after the students learned how to structure code in functions and procedures. At this point, it is common that the students have to implement a simple game or a text editor. The *Shame* gallery, then, will reveal code snippets with following problems: poor division of code into small subprograms, procedures that perform more or less instructions than the name suggests, procedures that access global variables instead of use parameters, bad code indentation, etc. In *Fame* gallery, there will be good code snippets, pointing to solutions that overcome bad solutions of *Shame*. Other example is teaching of functional programming with languages like Haskell and Lisp. Usually, it proposes to students to solve several kind of problems with lists, using recursive approaches and higher order functions. When students already known both alternatives, the *Shame* gallery could point, for example, code snippets that use explicit recursion instead of higher order functions as `map` and `filter` (since higher order functions enable use of lambda expression in several modern languages [Pierce 2002]).
- **Related patterns:** Application of HoFS supposes that students should produce code to implement some software, even an incomplete one. So, patterns related to teaching-learning can be used [Schmolitzky 2007]. In addition, the HoFS pattern is centered on the idea that students can benefit themselves of frequent feedback to advance and improve their programming skills. In this way, it can be complemented with other pedagogical patterns that focus in *continuum feedback* [Larson et al. 2008]. HOFS share some objectives of [Köppe et al. 2015] patterns to mitigate common issues in CS courses, like in ACTIVATING DELIVERY FORMS and DISCUSSION STATEMENTS. Other related patterns are ACTIVE STUDENT, SET THE STAGE and SUITABLE DELIVERY FORM SELECTION [Board 2012; Köppe et al. 2015]. HOFS also have several aspects in common with USE STUDENT'S SOLUTIONS [KÖPPE et al. 2015], since both patterns suggest the active discussion of snippets from students solutions. Additionally, HOFS explores the use of partial deliveries of assignments before its deadline to strenghen the engagement of students into the discussions.



(a) (b)

Fig. 1. Sample slides from the "Fame" (a) and "Shame" (b) galleries. Note the use of contrasting visual elements.

3. APPLICATION AND EVALUATION OF HOFS

We applied the proposed pattern to fourth semester students of a course which attend to Object Oriented Programming (OOP), offer by Federal University of Santa Maria, Brazil, to students of third and subsequent semesters of undergraduate course in Computer Science. The first contact of students with OOP–usually with Java

language–happens in this course. In the first application of HoFS, we perceive a good acceptance by students since: (i) students become involved during code exposition, paying more attention to teacher's talks and (ii) some students were concerned about their code been shown (or not) into some gallery.

Given these observations, we put the pattern in practice on the last course offer, following previously established procedures and criteria, aiming to collect data about patterns' impact on OOP learning. The remaining of this section refers to evaluation of application of HoFS.

We did not found any criteria or procedure to systematically evaluate pedagogical patterns in literature. Then, we decide to evaluate 2 aspects that, in our experience, are potential indicators about the efficiency (or not) of pattern. One of those is the students' **acceptance** of approach, measured by a survey that has been answered in course ending. The second is students' code **production**, which may contains good and/or bad programming practices, which was evaluated after each delivery. We understand that both aspects complement themselves. As an example of this complementation, a student can positively evaluate the experience, but the experience does not lead to any practical improvement in produced code.

3.1 Acceptance evaluation: methodology and results

To evaluate the acceptance of HoFS pattern, we prepare a 2 step survey, each one presented in form of items weighed in 5-point Likert scale, from -2 (totally disagree) to 2 (totally agree). The first step has 5 questions related to students' general perception about the pattern, relating it with their previous learning experiences. The second step has 4 questions about format adopted in exhibition and discussion of galleries, which comprises of interactive lectures using slides to expose code snippets, followed by on-line publication of a brief material for future reference (as presented in Section 2). Also, answers for second step of survey pointed some adjustments needed in pattern application.

The survey was anonymously answered by 15 students, in total of 22 students that attended the course until the end of semester. These students had practice and theoretical lectures, performed assignments and developed 2 individual projects with OOP. The HoFS patterns was applied in 3 times: (1) after delivery of first project, (2) in a partial delivery of second project (50% of requirements implemented) and (3) at deadline of second project. Students were evaluated by their performance to develop the software and by delivered report quality. The survey was applied at end of semester, after publication of scores.

We present the results of first step of survey in Figure (2). Note that one question of this part of survey has a negative proposition ("I did not liked to see my errors exposed, even anonymously"). In general, collected answers strengthen the initial hypothesis of good approach acceptance. Moreover, students perceived the originality of approach to disagree with "I already have lectures like this in other disciplines".

We present the results of second step of survey in Figure (3). In this regard, we observed a general acceptance of format, but some items need more attention: (1) Fame and Shame comments presented before assignment deadline were seen as more useful than those presented after deadline; (2) several students answered "Partially agree" to item "The published material for use outside of classroom was sufficient". The item (1) suggests that, when possible, the pattern should be applied while students are coding their solutions. The item (2) reveal a potential gap between effectiveness of galleries during and after lectures. The reasons of this can be investigated in new essays but, as a immediate alternative, we estimate that publication of videos about galleries' presentation can overcome those gap.

3.2 Production evaluation: methodology and results

To evaluate the impact of HoFS pattern in quality of students' productions, we considered 2 OOP projects assigned to students. These projects were developed in 1 and 3 weeks, respectively, and were delivered to teacher in two opportunities, as stated in previous section.

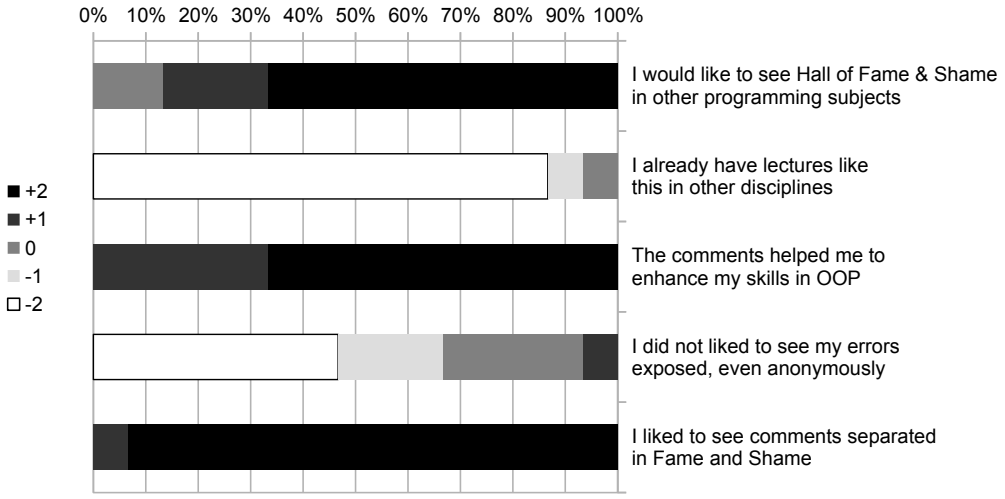


Fig. 2. Evaluation of general perception of pattern by students.

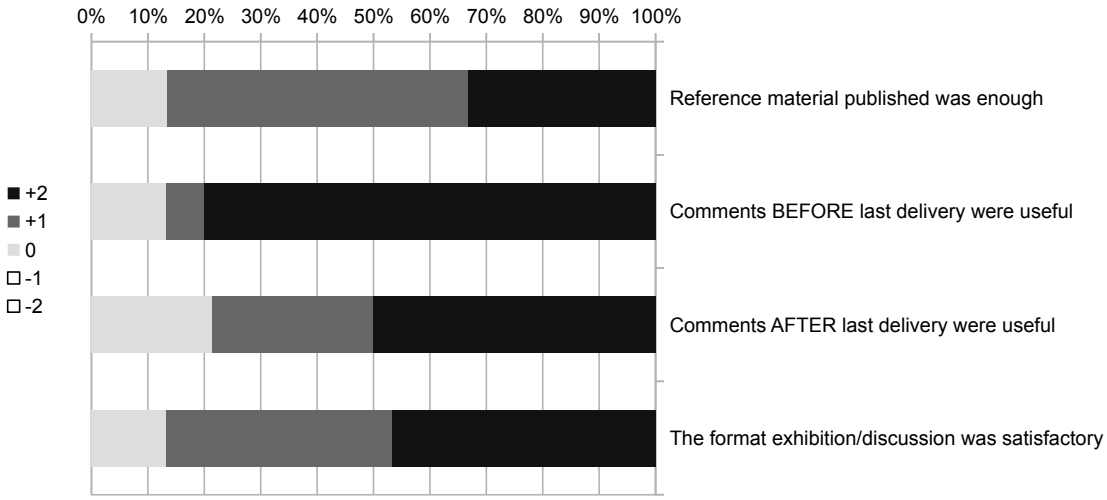


Fig. 3. Evaluation of format used no lectures following HoFS pattern.

The first project was a parking lot management software with a simulation of arrival and departure of vehicles. This project has no requirements about user interface, but emphasizes a object oriented implementation of domain in question. HoFS was applied after the deadline of this project.

The second project was about the development of a desktop graphical user interface, in Java, aiming to manage the records of fuel stations of a city with their respective pricing history. The project should use the MVC (Model-View-Controller) pattern and persist data in file or database. To this assignment, we applied HoFS pattern before and after project deadline.

For analysis of students' production, we consider all codes delivered by them on 3 previously mentioned moments, in a total of 17 and 20 projects delivered for the first and second assignments, respectively. For each

Fame and Shame exposition, we choose an average of 15 code snippets to present and discuss, all of them about good and bad practices of several aspects of OOP. The students could access the galleries in course web-page.

The subject of two projects allows use of optional good OOP approaches (as example, Data Access Objects for data persistence). By other side, some practices related to basement of OOP should be present in all works. In this situation, the hypothesis was that HoFS pattern can stimulate the students to correct their bad practices between a assignment to other. Thus, among all good and bad practices identified in projects, we choose 3 bad practices that their occurrences were counted before of first HoFS (before deadline of first project) and after of last HoFS (after deadline of last project). These practices were (1) code duplication rather than polymorphism, (2) violation of principle of single responsibility on class implementation and (3) public visibility of instance's attributes.

In Table (I), we present identified occurrences, being that a student can be responsible by more than one occurrence. We also can note that occurrences of bad practices decreased, as expected. It is well known that the nature of teaching-learning is very complex and, for this, we can not say undoubtedly that reduction of bad practices was only due to HoFS application. Nevertheless, we interpret these results as evidence about effectiveness of pattern. More important than this is the advances presented by the students and stimulated by HoFS.

Table I. Occurrences of bad and good coding practices after and before HoFS application.

Bad practice	Occurrences	
	Before	After
Code duplication	5	1
Single responsibility principle violations	10	2
Public visibility of instances attributes	3	0

4. FINAL REMARKS

In this work, we characterized a pedagogical pattern focused in programming teaching. This pattern is related to others of continuous feedback emphasis, but it differs since it is based essentially on analysis and exhibition of code snippets that represents good and bad programming practices.

The pattern was applied in a higher education institution and, on its last application, we performed an evaluation based in students acceptance of pattern and into analysis of code produced by themselves. The results show to efficiency and originality of pattern. Aiming to ease the reproduction of pattern in other time, galleries were published on-line and are available in `http://www-usr.inf.ufsm.br/~andrea/hofs-plop2016`.

We know that, a single pedagogical pattern can not guarantee the success of teaching-learning process. However, pedagogical patterns are an useful resource to teachers that want to improve their practices, as more patterns were documented, applied and discussed. Thus, as future prospects related to this work, we expect that more teachers share original patterns and/or report their application experiences of pedagogical patterns in Computation.

Lastly, the main idea behind some pedagogical patterns are generic and useful in several knowledge areas. As an example, the core of HoFS pattern is the analysis of trade-offs related to decision making, which can be extended to any design process. As a future work, the code snippets repository of previous applications of HoFS might be published together to the *fame* and *shame* galleries, extending our original approach to a "system of patterns".

Acknowledgement

The second author would like to thank CAPES foundation for his masters scholarship.

We would also to thank our shepherd Hironori Washizaki for his valuable comments. We are grateful for his patience during our (several) interactions and we have no doubt that his observations brought a lot of improvement to our work.

REFERENCES

Joseph Bergin. 2000. Fourteen Pedagogical Patterns. In *EuroPLoP*. 1–49.

Joseph Bergin, Jutta Eckstein, Markus Volter, Marianna Sipos, Eugene Wallingford, Klaus Marquardt, Jane Chandler, Helen Sharp, and Mary Lynn Manns. 2012. *Pedagogical patterns: advice for educators*. Joseph Bergin Software Tools.

Pedagogical Patterns Advisory Board. 2012. Pedagogical Patterns: Advice for Educators. *Joseph Bergin Software Tools* (2012).

Michael E Caspersen and Michael Kolling. 2009. STREAM: A first programming process. *ACM Transactions on Computing Education (TOCE)* 9, 1 (2009), 4.

Peter J. Denning. 1989. A Debate on Teaching Computing Science. *Commun. ACM* 32, 12 (Dec. 1989), 1397–1414. DOI:http://dx.doi.org/10.1145/76380.76381

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1993. *Design patterns: Abstraction and reuse of object-oriented design*. Springer.

Tony Jenkins. 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, Vol. 4. 53–58.

Christian Köppe. 2015. Towards a Pattern Language for Lecture Design: An inventory and categorization of existing lecture-relevant patterns. In *Proceedings of the 18th European Conference on Pattern Languages of Program*. ACM, 3.

CHRISTIAN KÖPPE, RALPH NIELS, ROBERT HOLWERDA, LARS TIJSMA, NIEK VAN DIEPEN, K Van Turnhout, and R Bakker. 2015. Flipped Classroom Patterns-Using Student Solutions. In *Preprints of the 22nd Pattern Languages of Programs conference, PLoP*, Vol. 15.

Christian Köppe, Michel Portier, René Bakker, and Stijn Hoppenbrouwers. 2015. Lecture Design Patterns: More Interactivity Improvement Patterns. In *Preprints of the 22nd Pattern Languages of Programs conference, PLoP*, Vol. 15.

Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin*, Vol. 37. ACM, 14–18.

Kathleen A Larson, Frances P Trees, and D Scott Weaver. 2008. Continuous feedback pedagogical patterns. In *Proceedings of the 15th Conference on Pattern Languages of Programs*. ACM, 12.

Yishay Mor, Harvey Mellar, Steven Warburton, and Niall Winters. 2014. *Practical design patterns for teaching and learning with technology*. Springer.

Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin* 39, 4 (2007), 204–223.

Benjamin C Pierce. 2002. *Types and programming languages*. MIT press.

Anthony Robins. 2010. Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education* 20, 1 (2010), 37–71.

Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: A review and discussion. *Computer science education* 13, 2 (2003), 137–172.

Axel Schmolitzky. 2007. Patterns for Teaching Software in Classroom.. In *EuroPLoP*. 37–52.

Helen Sharp, Mary Lynn Manns, Phil McLaughlin, Maximo Prieto, and Mahesh Dodani. 1996. Pedagogical patterns—successes in teaching object technology: a workshop from OOPSLA'96. *ACM SIGPLAN Notices* 31, 12 (1996), 18–21.

Elliot Soloway and James C Spohrer. 2013. *Studying the novice programmer*. Psychology Press.

Phit-Huan Tan, Choo-Yee Ting, and Siew-Woei Ling. 2009. Learning difficulties in programming courses: undergraduates' perspective and perception. In *Computer Technology and Development, 2009. ICCTD'09. International Conference on*, Vol. 1. IEEE, 42–46.

Brenda Cantwell Wilson and Sharon Shrock. 2001. Contributing to success in an introductory computer science course: a study of twelve factors. In *ACM SIGCSE Bulletin*, Vol. 33. ACM, 184–188.

Leon E Winslow. 1996. Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin* 28, 3 (1996), 17–22.