Using Software Engineering Design Principles as Tools for Freshman Students Learning

Ivan Cabezas
LIDIS
School of Engineering
Universidad de San Buenaventura
Cali, Colombia
imcabezas@usbcali.edu.co

Patricia Caratozzolo
School of Engineering
Campus Santa Fe
Tecnologico de Monterrey
Ciudad de Mexico, Mexico
pcaratozzolo@tec.mx

Rocio Segovia
LIDIS
School of Engineering
Universidad de San Buenaventura
Cali, Colombia
ersegovia@usbcali.edu.co

Eileen Webb
President
Accreditation Preparation LLC
ewebb@AccreditationPreparation.com

***Abstract*— In this innovative practice work-in-progress paper, the software engineering design principles that are used for Programming Fundamentals, and Object-Oriented Programming courses, within a -recently renewed curriculum- software engineering program, are reviewed and discussed as tools to introduce first-year students into the engineering design skills learning pathway of the Engineering Accreditation Commission of the *Accreditation Board for Engineering and Technology*, *ABET*. Special attention is paid to highly abstract concepts such as high cohesion, low coupling, *Keep it Short and Simple, Don't Repeat Yourself, Single Responsibility Principle*, among other object-oriented design principles. The aim of such exercise is two-fold: (i) to address how design principles can be used to guide students' learning process, and (ii) to support pedagogical decision making for the software engineering program. The outtake of the whole research planned by the authors is as follows. On the one hand, design principles are discussed and analyzed, from the first-year programming courses perspective, aiming a pedagogical path planning for student outcomes attainment within a Project-Based-Learning approach. On the other hand, the discussion on how the program takes advantage of assessment and evaluation processes, can be useful for programs and faculty concerned with fulfilling the continuous improvement criterion, as defined by *ABET*. The presented work-in-progress focuses on reflection about design principles and establishing a link among them, with the engineering design process. In this way, this paper separates from similar works that may focus on design principles by itself, without the mentioned intentionality. Reported qualitative results are focused on conducted experiences so far within our software engineering program.**

Keywords*— *design principles, software engineering, first year courses, learning pathway.

I. Introduction

The Engineering Accreditation Commission (EAC) of the Accreditation Board for Engineering and Technology (ABET), introduced several changes to the accreditation criteria during the 2019-2020 cycle. Some of them are focused on the engineering design process. An explicit and thorough definition of it was provided, and the former student outcome c was restated in the student outcome 2, focusing on applying the engineering design process to produce solutions meeting needs while considering multiple factors. Such changes highlight how relevant it is to lead engineering students into learning pathways for developing engineering design skills. Faculty from a software engineering program have realized how challenging it is for first-year students to properly develop engineering design skills. In practice, there is an overlap between the engineering design process as defined by the ABET's EAC, and the software engineering design principles, but they are not quite the same. The former is broader as it includes identifying opportunities, requirements definition, considering risks, and generating and evaluating solutions, among others.

Engineering problems, on which software-intensive systems are required as solutions, can be recognized as complex problems due to two main factors: involving a diverse group of stakeholders and include many components or non-trivial sub-problems [1]. Software systems have become pervasive in our society and mediate in almost every aspect of modern life [2]. However, in practice, the high demand for skilled software and computer science engineers is correlated to a high dropout of such program's students, mainly during the first year [3, 4]. The sometimes-overwhelming complexity of programming paradigms and the emotions related to programming learning can be among the challenges faced by first-year students, impacting high dropouts ratios [5]. Consequently, all research efforts devoted to addressing issues arising, mainly during the first year of software and computer science engineering programs, can be useful to engineering schools and their faculty [6].

This study presents an innovative *work-in-progress* focused on introducing first-year students into the ABET's EAC engineering design skills learning-pathway. We reviewed the design principles [7] that can be used to align the structured and Object-Oriented Programming (OOP) paradigms with the ABET's EAC engineering design skills. We discuss how the reviewed design principles can be linked through the transition from the structured to the object-oriented paradigm and explain how such discussion has supported our program in its continuous improvement effort. Finally, we present qualitative results obtained so far and highlight the next phases of our ongoing research. The paper is structured as follows: Section II briefly summarizes the necessary background. Section III describes the innovative

practice approach. Preliminary qualitative results are presented in Section IV. Section V closes the paper with final remarks and future work.

II. Background

A design principle is, in brief, and from the authors' perspective, a general guideline supporting software engineers to build a maintainable software system. The characteristic of being maintainable is related to the degree of effectiveness and efficiency with which a software product or software system can be modified to improve it, correct it, adapt it to changes in the environment, and requirements [8]. Maintainability also implies the following sub-characteristics: modularity, testability, modifiability, analysability, and reusability. These sub-characteristics are summarized in Table 1.

TABLE 1. Sub-characteristics of Maintainability

MODULARITY
Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.
TESTABILITY
Degree of effectiveness and efficiency with which test criteria can be established for a system, product, or component and tests can be performed to determine whether those criteria have been met.
MODIFIABILITY
Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
ANALYSABILITY
Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.
REUSABILITY
Degree to which an asset can be used in more than one system, or in building other assets.

It is worth highlighting that a design principle is not the same as a design pattern since the latter can be understood as a reusable solution to a commonly occurring problem within a given context. For instance, factory and singleton are OOP patterns [9], while increasing cohesion and decreasing coupling is a design principle. In other words, understanding coupling and cohesion as attributes that summarize the degree of interdependence or connectivity among subsystems and within subsystems, respectively [10], focuses first-year students on achieving a highly cohesive and lowly coupled software system, as a design goal. Moreover, design principles are inherently related, while design patterns are task-specific, or even concrete examples of a specific design principle. The main design principles used in the conducted innovative practice, beyond cohesion and coupling are: Keep it Short and Simple (KISS), Don't Repeat Yourself (DRY), Single Responsibility Principle (SRP), Open-Close Principle, and Liskov Substitution Principle (LSP). This *work-in-progress* paper will focus only on KISS, DRY and SRP.

KISS. The origin of the KISS principle is associated with an aircraft engineer aiming to obtain a product that could be repaired in a combat field, requiring basic training and a few simple tools. It was originally formulated as Keep It Simple Stupid, but such connotation is deliberately avoided within the conducted innovative practice classroom experience. The KISS principle can be understood as a usability principle, looking at simplicity as a design goal. An equivalent formulation of the KISS principle is stated in a quote by the French writer Antoine de Saint Exupéry: "It seems that perfection is attained, not when there is nothing more to add, but when there is nothing more to take away.". In an example of how the design principles are inherently related, the KISS principle can also be understood as obtaining a software product with fewer modules and lesser interconnections among them. The KISS principle impacts on analysability and testability sub-characteristics.

DRY. The DRY principle is formally stated in [11] as "Every piece of knowledge must have a single, unambiguous, and authoritative representation within a system.". It aims to avoid duplication or redundancy of common functionality, by saying anything in a software piece once and only once. When the DRY principle is properly applied, a software module modification does not imply a change in another logically unrelated element. The DRY principle relates to modularity and reusability sub-characteristics.

SRP. The SRP stands for the S in the S.O.L.I.D. mnemonic acronym of OOP design principles. However, the SRP is not applicable only in the OOP. The SRP states that each software module should have one and only one reason to change. It is related to the separation of concerns, cohesion (among modules that change for the same reasons) and coupling (among modules that change due to different reasons). The SRP impacts on modularity and modifiability sub-characteristics.

Regarding the engineering design process for devising a system, component, or process, it is defined by the ABET's EAC as applying basic sciences, mathematics, and engineering sciences, during an iterative decision-making process, to convert resources into solutions, according to desired needs, specifications and within constraints. It involves developing requirements, performing analysis and synthesis, generating multiple solutions, evaluating solutions against requirements, considering risks, and making trade-offs, to obtain a high-quality solution under the given circumstances, among others.

III. Pedagogical Framework

Different cognitive theories will be used in this study, specially adapted for engineering students. Firstly, the theory of cognitive functioning of Jerome Bruner for students' development that deals with students' ability to develop new technological products through the conversion of the imaginative concepts into a dependable reality [12]. Secondly, the theory of mind development by John Dewey, integrating education across subjects, and engaging students with real-world applications to form their proper judgments and to increase their practical knowledge [13]. Additionally, Ennis's adaptation of the conceptual contribution of critical thinking was used considering that creative thinking, like critical thinking, is characterized by three principles [14]: Background knowledge is essential for creative thinking in a given domain; Transfer of creative thinking between domains

becomes possible when accompanied by an instruction for such transfer; and finally, strong or even moderate specificity instruction is likely to be more effective than general creative thinking instructions in the form of subsequent add-ons.

IV. Innovative Practice Approach

The presented innovative practice was conducted in the context of a software engineering program, with an adapted curriculum from ten to nine semesters, and started to be offered during the 2019 first semester. It comprises two programming courses during the first year: Programming Fundamentals, and Object-Oriented Programming. Both of them pursuit supporting students' attainment of ABET's EAC student outcomes number 2 and number 5, follow a project-oriented learning approach, and share an iterative and incremental software development methodology (involving analysis, planning, coding, and testing cyclical phases), with adjusted deliverables according to course goals and content. The implications and expectations, within the courses' context of the discussed design principles, are shown in Table 2. The selection and use of presented approach is the students' first introduction to engineering design as an iterative decision-making process. Programming Fundamentals course foster knowledge and skills related to problem-solving and structured programming, while the Object-Oriented Programming course addresses the OOP paradigm and design patterns. C++11 was chosen as the main language to be used in both courses. However, a presented key concept, is also commonly illustrated in languages such as Processing [15], Python 3.7 [16], and C# [17]. In this way, the chosen language is merely seen as a tool. In both courses, students follow and document the software development methodology. The first course's software pieces can be described as a single .cpp file, involving several functions, and avoiding explicitly using global variables. The second course's software pieces involve multiple .cpp and .h files, (i.e., classes) following discussed design patterns. In both cases, students should also document the decisions made regarding how design principles were fulfilled during the engineering design process.

TABLE 2. Implications and expectations of discussed design principles, within the courses' context

KISS	
Programming Fundamentals	*Object-Oriented Programming*
This is the first presented design principle. It enforces a clear coding style: using adequate and meaningful variables and functions names, indentation, removing useless code, and introducing non-trivial documentation. It also allows introducing the refactoring concept between development methodology iterations. Students are faced with analyzing and modifying code developed by classmates to solve a similar or the same problem.	In this course context, the KISS principle aims to avoid unnecessary complexity in the proposed solution, and at the same time, allows to exemplify how software reuse increases productivity. It also motivates discussions about the consumed resources (i.e., time and space) by a specific solution.

DRY	
Programming Fundamentals	*Object-Oriented Programming*
This principle allows us to naturally introduce the concept and the convenience of using functions to replace multiple appearances of the same purpose code (e.g., reading the same type of input). It also allows introducing the most basic software reuse approach, not only using software libraries but also among different software pieces, to not coding twice the same functionality.	The software reuse goal allows introducing relations between classes, as well as to discuss cohesion and coupling concerns. Additionally, it supports and motivates the use of design patterns to avoid rethinking on how to approach an initial design or how to tackle specific issues of the proposed solution. It opens the door to address OOP characteristics, such as inheritance and polymorphism. Regarding teamwork, it puts on the table the necessity of cooperating, explicitly discussing design ideas and assigning tasks, to avoid inter-developer duplications. In that scenario, another meaning to the DRY principle is an understanding of the no need for reinventing the wheel, as the OOP has a lot to offer, in terms of productivity, only by message passing to already available libraries' classes.

SRP	
Programming Fundamentals	*Object-Oriented Programming*
In this context, a function is considered as a module. Consequently, a function should resolve a single functionality, instead of being a potpourri of tasks. This notion reinforces the goal of properly structuring a piece of software by combining a set of building blocks, in a trade-off between general and reusable modules and problem-specific modules, within common and single-purpose layers.	The SRP is used as a way to pursue high cohesion, and motivates the use of design patterns. It opens the door to address encapsulation and abstraction issues, as well as the necessity of having relations between classes.

High cohesion and low coupling	
Programming Fundamentals	*Object-Oriented Programming*
Cohesion and coupling are deliberately not explicitly addressed in this course since, based on previous experience, we found them as highly abstract concepts for students working on a structured but single piece of software (i.e. a single .cpp file). However, it is implicitly addressed by the SRP.	High cohesion and low coupling are the first design principles, or goals, addressed in the course, which allows introducing the above-mentioned principles as decision-making tools and guidelines. Applying the SRP unavoidably leads to address coupling. So, pursuing a trade-off between cohesion and coupling during the engineering design process arises naturally. Obtaining a lowly cohesive and highly coupled design is identified as a risk for the design process, impacting on software maintainability.

Engineering design and teamwork. Taking advantage of the Project-Based-Learning approach, a problem, and its solution evolve during the course by adapting constraints and requirements. In this way, maintainability is a specific concern during the pedagogical experience delivered by the courses. Feedback is provided to teams, during sessions on which design decisions are synthesized and communicated, in order to expose how design principles were followed during the design process, how the team roles were assigned, and concluding about how properly the presented solution fulfills requirements and constraints. The software engineering program relies on the project-oriented learning approach, as it fully engages students in their knowledge construction, collaboratively. It also offers advantages for later courses (from sophomore and beyond), which apply a flipped classroom environment. Consequently, it is also convenient for first-year students, strengthening their project-based learning skills.

V. Methodology

Design. The design chosen for the project will focus on experimental, quantitative, Four-Group Solomon-type [18]. The aim is to control the possible interaction that could exist between the Pre-Test and the Treatment. This design will allow the results to be generalized also for students who do not receive the Pre-Test. Two groups will receive the Pre-Test, and two others will not; two groups will receive Treatment, and two others will not. The comparison between groups will be the fundamental element of the investigation: we will work with two groups, the one that will receive the Treatment (Experimental Group) and the one that will not receive it (Control Group). Students will be assigned to groups randomly.

Process. We think that the quantitative-experimental research methodology will be the most appropriate to establish causal relationships between groups of variables. The independent variable will be our experimental variable, and the dependent variable was our result or criterion, with which we intend to achieve the effects expected in the study.

Instrumentation. Different types of instruments will be used for both Pre-Tests and Post-Tests. We are evaluating the use of rubrics based on the adaptation of existing rubrics, the VALUE Rubrics, from AAC & U, developed for Essential Learning Outcomes of the Association of American Colleges and Universities [19].

VI. Preliminary Results

In the current *work-in-progress* research phase, we have identified the below-mentioned issues in each course. As the research keeps moving on, we will choose strategies and approaches for quantitatively estimating prioritized concerns and elaborate continuous improvement plans based on those findings.

A. Programming Fundamentals

Found proper and effective ways of engaging students in a first-semester programming course has been historically challenging. Follow design principles are not the students' highest priority, as is *make it work:* develop a software piece that solves the problem (i.e., produces the expected output for a specific input). Only at the final half of the course, students seem to show a better comprehension and strictness to the SRP, which turn to impact on the adherence of KISS and DRY positively. The DRY principle is also challenging to them, since there is a trend, on most students to tackle each problem from scratch, without considering what can be reused from previous exercises. Regarding teamwork, it is quite common the case on which they choose to work alone. On this matter, authors have explored alternatives such as randomly form groups or assign software development roles to different team members. So far, our perception is that teamwork behavior is not only a matter of this course since it seems students tend to form stable groups among different courses.

B. Object-Oriented Programming

Authors have identified a shocking first impression on students as this course strongly emphasizes design tasks and concerns. Students seem to perceive the use of the OOP theory as an arbitrary increase in software development complexity. It may be due to the size of their programs is still relatively small, so they do not realize the advantages of using the OOP paradigm at a glance. In terms of cohesion and coupling, students show a proper apprehension, despite their solutions tend to be more coupled than expected. In practice, achieving a highly cohesive and lowly coupled software product arises as a challenge to students, on which its mere definition is not so useful. The SRP is also difficult to follow and more important to them than DRY, as their designs presentations reflect. The KISS principle is challenging to students since they rely on examples found on the internet, perhaps without full comprehension. In that way, the obtained product can often be described as a sledgehammer to crack a nut. Regarding teamwork, students show a more mature attitude and increased willingness and skills to promote a collaborative environment.

VII. Final Remarks and Future Work

In their latest reports, the Organization for Economic Co-operation and Development, OECD, and the World Economic Forum, WEF, introduced a comparison of today's skills with those demanded of future professionals to face the Fourth Industrial challenges Revolution [20,21].

The five skills expected to be trending by 2022 are Analytical thinking and innovation; Active learning and learning strategies; Creativity, originality, and initiative; Technology design and programming, and Critical thinking. One of the most important conclusions is that in a few years, there will be types of work that will require competition in new skills that have not been part of the basic skill set of that occupation previously. In this regard, we believe that preparing students for design skills will help them acquire and strengthen the skills mentioned above. Moreover, we think some approaches already used to assess critical thinking can be applied to our research based on the text and documents students produce to explain their adherence to design principles [22].

So far, the discussions and the subjective findings have allowed us to introduce changes in the OOP course weekly schedule to achieve a better comprehension of the OOP paradigm. Changes have also been introduced at the final part of the Programming Fundamental course to promote a smooth transition between courses. We have found interesting and engaging to rely on second-semester students to present a problem-solving exercise, showing both approaches, structured vs. object-oriented, highlighting the design principles that followed during the engineering design process.

A current challenge found by our research is the transition from a presence learning & teaching scenario to a remote environment due to the COVID-19 quarantine and isolation measures dictated by the national government. The authors should also consider which of previously identified assessment methods are still suitable for the current teaching and learning experience.

Acknowledgment

The first author would like to acknowledge the Universidad de San Buenaventura-Cali for supporting this research.

The second author would like to acknowledge the technical support of Writing Lab, TecLabs, Tecnologico de Monterrey, and the financial support of Novus Grant with PEP no. PHHT090-19ZZ00008, TecLabs, Tecnologico de Monterrey, in the production of this work.

REFERENCES

[1] ABET Inc., Abet.org, 2020. [Online]. Available: https://www.abet.org/. [Accessed: 18- Apr- 2020]

[2] C. Becker et al., "Sustainability Design and Software: The Karlskrona Manifesto", 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, 2015, pp. 467-476.

[3] P. Schoeffel, R. S. Wazlawick, V. F. C. Ramos, A. Vahldick and M. d. Souza, "Identification of Pre-University Factors that Affect the Initial Motivation of Students in Computing Programs: A multi-institutional case study," 2018 IEEE Frontiers in Education Conference (FIE), San Jose, CA, USA, 2018, pp. 1-8.

[4] B. Standl, E. Wetzinger, G. Futschek and E. A. Guenther, "First-year computer science students' perception of lectures in relation to type of high-school education", 2018 IEEE Global Engineering Education Conference (EDUCON), Tenerife, 2018, pp. 1389-1394.

[5] R. Colomo-Palacios, T. Samuelsen and C. Casado-Lumbreras, "Emotions in Software Practice: Presentation vs. Coding", 2019 IEEE/ACM 4th International Workshop on Emotion Awareness in Software Engineering (SEmotion), Montreal, QC, Canada, 2019, pp. 23-28.

[6] N. Bosch, S. D'mello, and C. Mills, "LNAI 7926 - What Emotions Do Novices Experience during Their First Computer Programming Learning Session?," 2013.

[7] W. Haoyu and Z. Haili, "Basic Design Principles in Software Engineering", 2012 Fourth International Conference on Computational and Information Sciences, Chongqing, 2012, pp. 1251-1254.

[8] ISO 25000 [Online] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/ [Accessed: 18- Apr- 2020]

[9] J. Bosch, "Specifying frameworks and design patterns as architectural fragments", Proceedings Technology of Object-Oriented Languages. TOOLS 27 (Cat. No.98EX224), Beijing, China, 1998, pp. 268-277.

[10] E. B. Allen and T. M. Khoshgoftaar, "Measuring coupling and cohesion: an information-theory approach", Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403), Boca Raton, FL, USA, 1999, pp. 119-127.

[11] A. Hunt and D. Thomas, "The Pragmatic Programmer: From Journeyman to Master", Addison-Wesley Professional, 1999.

[12] Bruner, J. S. *Actual minds, possible worlds*. Harvard University Press, 2009.

[13] Dewey, John. *How we think*. Courier Corporation, 1997.

[14] Ennis, R.H. Critical thinking and subject specificity: Clarification and needed research. *Educational researcher*, *18*(3), 1989. pp.4-10.

[15] Processing [Online] https://processing.org/ [Accessed: 18- Apr- 2020],

[16] Python 3.7 [Online] https://www.python.org/ [Accessed: 18- Apr- 2020]

[17] C# [Online] https://docs.microsoft.com/en-us/dotnet/csharp/ [Accessed: 18- Apr- 2020]

[18] Cohen, Louis, and Lawrence Manion. *Método de investigación educativa*. No. 37.012. La Muralla, 1990.

[19] VALUE Rubrics. Association of American Colleges & Universities. https://www.aacu.org/value-rubrics. Writer's Handbook. Mill Valley, CA: University Science, 1989.

[20] K. Howells. "The future of education and skills: education 2030: the future we want.", 2018.

[21] World Economic Forum Boston Consulting Group (BCG). "Towards a reskilling revolution: a future of jobs for all", World Economic Forum, Geneva, Switzerland, 2018.

[22] P. Caratozzolo, A. Alvarez-Delgado, and S. Hosseini, "Strengthening critical thinking in engineering students". Int J Interact Des Manuf 13, 995–1012 (2019).